# Optimal Sorting of Rolling Stock

Von der Carl-Friedrich-Gauß-Fakultät
Technische Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades

Doktor der Naturwissenschaften
(Dr. rer. nat.)

**genehmigte Dissertation**

von

Dipl.-Math. Ronny Stefan Hansmann

geboren am 26. April 1978 in Rabenstein

| | |
|---|---|
| Referent: | Prof. Dr. Uwe T. Zimmermann |
| Korreferent: | Prof. Dr. Alexander Martin |
| Eingereicht am: | 13. Oktober 2010 |
| Mündliche Prüfung am: | 10. Dezember 2010 |

Für P.

# Zusammenfassung

Die vorliegende Dissertation beschäftigt sich mit dem optimalen Sortieren von Objekten, insbesondere von Güterwagen in Rangierbahnhöfen. Motiviert wurde diese Arbeit durch ein BMBF-gefördertes Projekt mit der BASF, The Chemical Company. Im Ludwigshafener Stammwerk der BASF wird ein Großteil der internen Logistik auf der Schiene abgewickelt. Ein Nadelöhr stellt dabei der werkseigene, kostenintensive Rangierbahnhof mit Ablaufberg dar, in dem eingehende Züge zerlegt und die Wagen zu Ausgangszügen umsortiert werden. Das Projektziel war es, ein Optimierungstool als Entscheidungsunterstützung für die Disponenten zu entwickeln, das vollautomatisch optimale Pläne zur Sortierung der Wagen erzeugt.

In diesem Kontext des Sortierens von Objekten wird eine umfassende Klassifizierung zahlreicher Varianten eingeführt. Die grundsätzliche Anforderung aller Varianten ist, eine eingehende Sequenz von Objekten in einer Sortieranlage möglichst optimal – d. h. in der Regel durch eine minimale Anzahl an zulässigen Operationen – in eine Ausgangssequenz zu überführen, welche eine im Vorfeld gewünschte Struktur aufweist.

In dieser Arbeit werden zahlreiche Varianten mathematisch formuliert. Für viele Varianten wird deren Äquivalenz zu bestimmten Graphenfärbungs-, Scheduling- sowie Bin-Packing-Problemen gezeigt. Die Dissertation beinhaltet eine komplexitätstheoretische Einordnung einiger Varianten. Für mehrere als theoretisch *schwer* bewiesene Fälle werden schnelle approximative Algorithmen vorgeschlagen, die Lösungen mit einer beweisbaren Güte liefern. Desweiteren werden neben heuristischen Methoden auch exakte Verfahren zur Bestimmung optimaler Lösungen vorgestellt. Unter anderem handelt es sich bei den eingesetzten exakten Ansätzen um LP- sowie Lagrangebasierte Branch-and-Bound-Verfahren, die auf verschiedenen binären Modellen beruhen. Die Lösungsmethoden werden durch die Auswertung von Rechenergebnissen für reale Daten evaluiert. Eine wesentliche Erkenntnis ist, dass sich für das reale Optimierungsproblem bei

der BASF mit den vorgeschlagenen Ansätzen innerhalb weniger Minuten optimale Pläne bestimmen lassen.

Den Abschluss der Dissertation bildet eine Kompetitivitätsanalyse diverser Online-Varianten, die dadurch gekennzeichnet sind, dass nicht alle relevanten Informationen zu Beginn der Planung vorliegen.

Abschließend sei auf das Verwertungspotenzial der in dieser Arbeit vorgestellten Optimierungsverfahren innerhalb anderer Anwendungsbereiche, in denen Sortieren, Stapeln, Lagern oder Verstauen eine Rolle spielen, hingewiesen.

# Acknowledgements

THIS thesis could not have been written without the aid and encouragement of many people. First of all, I would like to express my heartfelt gratitude to my supervisor Uwe Zimmermann for giving me the chance to work on this very exciting topic, and for providing me with steady support and freedom to pursue research directions in which I have been interested. I very much appreciate to benefit from his skills of applying Operations Research and Discrete Optimization to real-life problems, and his views on life in general. Furthermore, I wish to convey my sincere thanks to Alexander Martin for agreeing to co-referee this thesis.

The work would have been impossible without the financial support through a joint project "*Zeitkritische Ablaufbergoptimierung in Rangierbahnhöfen*" – together with BASF, The Chemical Company, Ludwigshafen – granted by the German Federal Ministry for Education, Science, Research, and Technology (BMBF). I am very grateful for the many motivating and supporting practical discussions as well as the real world data made available by Holger Schmiers, Matthias Ostmann, and Rainer Scholz (BASF, Service Center Railway). Moreover, I am obliged to Anna Schreieck and Josef Kallrath (BASF, Scientific Computing) for their support and encouragement.

Many thanks to all staff members of MO for providing a supportive and friendly working environment. I very much enjoyed our coffee with cake breaks and our discussions on soccer. I am particularly grateful to Stefan Krause and Thomas Rieger for proof-reading the manuscript.

To my dear parents: *Mutt, Dad, seht diese Arbeit als Dank für Euer Vertrauen und Eure Unterstützung!*

And last but surely not least, my thanks and my heart go to the two most important females in my life.

RONNY HANSMANN

# Contents

CHAPTER 1

# Introduction

THIS thesis is concerned with the problem of optimally rearranging objects, in particular, railcars in a rail yard. The work is motivated by a research project[1] of the Institute of Mathematical Optimization at Technische Universität Braunschweig, together with our practical partner BASF, The Chemical Company, in Ludwigshafen – in the following BASF for short.

For many *versions* (variants) of such rearrangement problems – including the real-world application at BASF – we state the computational complexity, we present optimization methods for determining schedules that are either optimal or close to optimal, and we discuss computational results from both a theoretical and practical point of view.

In addition to the railway industry, there are other fields of application in which efficiently rearranging, sorting, or stacking is an important issue. For instance, it is indeed conceivable that the results obtained in this thesis could be applied to solving certain piling problems in warehouses or container terminals.

Before introducing a classification system for a multitude of interesting versions, we briefly review the historical development of the methodological approach to rearranging trains and railcars in rail yards.

[1]This research was partially funded by the German Federal Ministry for Education, Science, Research, and Technology (BMBF) under grants no. 03-ZINJBS

## 1.1 Historical Review on Methods for Rearranging Railcars

Undoubtedly, one of the most essential achievements in engineering was the development of steam-powered engines at the end of the 18th century. This invention became a driving force behind the Industrial Revolution, underpinned increases in production capacity in many industries and gave birth to the railways, a fast and cost-efficient transportation system which lent additional impetus to the industrial age.

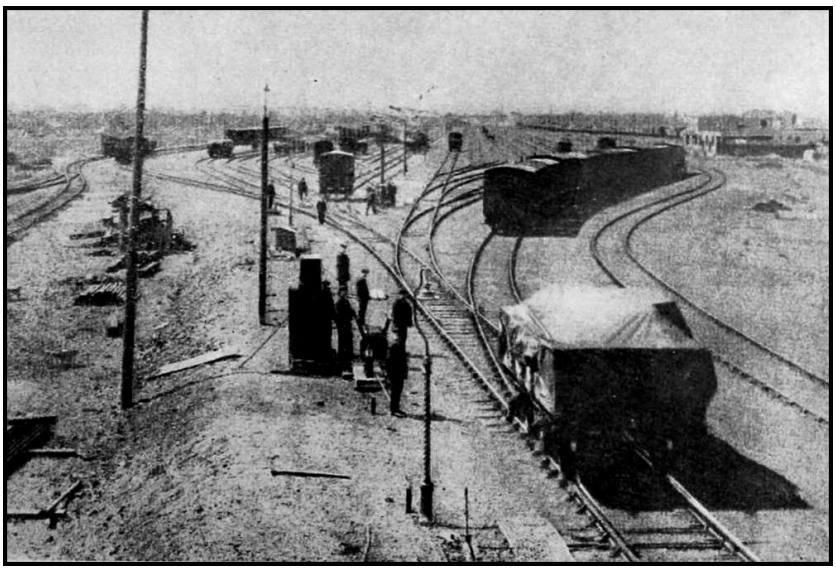

**Figure 1.1.** Feltham Marshalling Yard, England. *Source:* The New Zealand Railways Magazine, Volume 1, Issue 9 (February 25, 1927)

Rearranging the railcars was – and still is – one of the biggest challenges in operating railways. In addition to switching locomotives from one side of the train to the other – trains mostly shuttled between two stations in the early days – shunting was unavoidable in the case that a broken or malfunctioning railcar had to be replaced. Until the mid-19th century such rearrangements were performed at common rail stations where passengers could occasionally witness the effort workers devoted to this dangerous task. The public stations became operational bottlenecks as traffic increased; consequently, non-public *rail yards* (in other words *shunting yards*, *classification yards*, *marshalling yards*) were built, see Figure 1.1. Nowadays in these yards many inbound trains are split up, rearranged, and attached to several outbound trains

at the same time.

Futhner's method is one of the oldest methods for rearranging railcars in a rail yard. According to the authors of IVIĆ ET AL. (2007) it is named after the author Harry Futhner who in 1880 was the first to apply it in practice at the Liverpool station consisting of parallel dead-ended tracks. The task was to rearrange an incoming sequence of railcars – possibly a few trains in succession – in order to form $g$ outbound trains. For each railcar it was determined in advance with which train it had to leave, and the trains were required to depart from the station in a given order. Futhner's method is a two-step sorting procedure which requires $\lceil\sqrt{g}\rceil$ tracks. In the first phase the railcars of the 1st, $(\lceil\sqrt{g}\rceil+1)$-th, $(2\cdot\lceil\sqrt{g}\rceil+1)$-th, ... departing trains are placed on track 1, the railcars of the 2nd, $(\lceil\sqrt{g}\rceil+2)$-th, ... on track 2, and so on, see Figure 1.2. This classification allows – after pulling out all railcars in the order of increasing track numbers – a second sorting of the railcars to tracks, such that the trains can leave the station without additional rearrangements. This method presumably worked well in practice, since only a couple of trains had to be formed at the same time, i. e., $\lceil\sqrt{g}\rceil$ did usually not exceed the number of available tracks, and because the outbound trains carried only a few railcars such that the tracks were long enough for the implementation of Futhner's scheme.

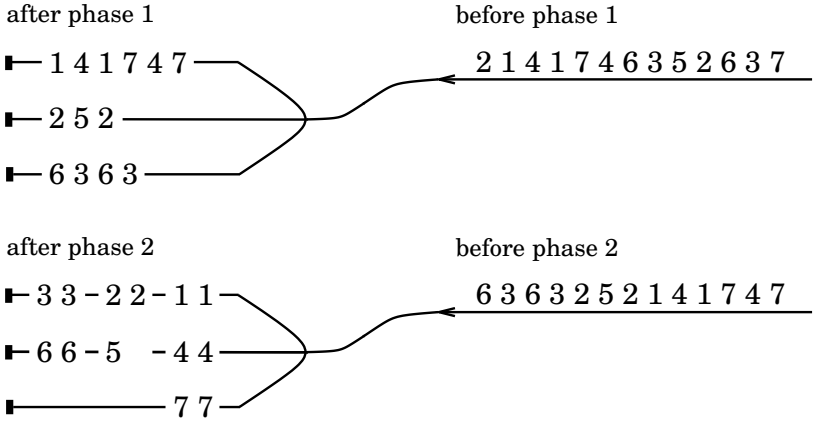


**Figure 1.2.** Futhner's method applied in Liverpool Station around 1880 (each number $i$ corresponds to a railcar that has to leave with train $i$)

Over the course of time, similar rule-based methods for rearranging railcars were developed and applied in practice. Among the most famous are the *simultaneous*, *triangular*, or *geometric* schemes. The

first articles discussing the benefits and drawbacks of various schemes – FERTIG (1927), WÖCKEL (1949), GRASSMANN (1952), FLANDORFFER (1953), BAUMANN (1959), PENTINGA (1959), KRELL (1962, 1963) – were published in the railways magazines. On the one hand, the advantage of these rule-based methods is their simplicity and transparency; out of habit the staff know exactly what to do, no matter how the arriving railcars are actually ordered. On the other hand, by not exploiting this particular order one may lose some potential for saving time and money. In other words, the sorting might be realized faster with less tracks and shunting operations with a scheme tailored to the incoming sequence of railcars. The oldest of such strategies found in the literature – see KÖNIG & SCHALTEGGER (1967), SCHALTEGGER (1967) – was introduced by the Group Operations Research at Schweizer Bundesbahnen in the late sixties of the last century. Under the direction of the mathematician Peter Schaltegger, they developed a mathematical optimization approach and implemented an algorithm in FORTRAN IV. Although they could – under certain assumptions – determine an optimal simultaneous scheme for the predicted order of incoming railcars on an Univac 1107 within seconds, there were obstacles for applying it in daily action. The problem of instantly giving the details of the computed schedule to all employees involved in the process was only one reason why optimization methods could not prevail against rule-based schemes in practice at that time. The next three decades produced little methodological progress for rearranging railcars, and from a theoretical perspective the literature – SIDDIQEE (1972), TARJAN (1972), PETERSEN (1977a,b), ASSAD (1981, 1983), DAGANZO ET AL. (1983), DAGANZO (1986, 1987b,a) – was again mainly on analyzing the effectiveness of rule-based strategies for different scenarios.

In the course of time various technical advances in rail yards created the prerequisites for convenient application of automatically generated schedules that guarantee an efficient rearrangement of specific incoming railcars. Automatic switches and brakes replaced mechanical ones. Nowadays the dispatcher most often monitors and controls the processes from the control tower, and the few people who are physically involved are connected via fast modern communication networks.

As a consequence, there is increasing interest on the practitioners side for active optimization tools that can automatically generate schedules. Before the turn of the millennium, it was rather rare for practitioners and researchers to work together in this field. However,

in recent years, quite a few projects between rail operators and universities were launched. Once inspired by the application, the involved researchers were intrigued by the beauty of the underlying theoretical problems. The following selection of recent publications shows that rearranging railcars has been a hot topic from both the theoretical and practical perspectives for the last decade: WINTER (2000), DAHLHAUS ET AL. (2000b,a), LÜBBECKE & ZIMMERMANN (2000), WINTER & ZIMMERMANN (2000), CORNELSEN & DI STEFANO (2004, 2007), DI STEFANO & KOČI (2004), FRELING ET AL. (2005), LÜBBECKE & ZIMMERMANN (2005), KROON ET AL. (2006), JACOB (2007), JACOB ET AL. (2007), HANSMANN & ZIMMERMANN (2008), CESELLI ET AL. (2008), MÁRTON ET AL. (2009), EGGERMONT ET AL. (2009), BORNDÖRFER & CARDONHA (2009), HAUSER & MAUE (2010). Relevant details and results of above publications are given at appropriate places throughout the thesis. For a recent introductory survey, see GATTO ET AL. (2009).

Nevertheless, in most railway operating companies there still exists no active optimization tool as decision support for the dispatchers at the present time. One reason may be that it is hard to come up with a standard approach. The schedules that need to be generated depend highly on the particular infrastructure of the rail yard, the configuration of inbound and outbound trains, and the requested objective. Thus, methods for computing schedules of high quality have to be tailor-made to the actual situation.

In this thesis we introduce a thorough classification of many versions of such rearrangement problems. Regarding optimization methods and computational complexity, we summarize known results and present new findings for a multitude of versions. In particular, we discuss the results obtained in our research project with BASF.

## 1.2 Classification of Rearrangement Problems

In rail yards incoming freight or passenger trains are split up, parked, and rearranged according to destination or according to railcar construction type, see Figure 1.3. Uncertain arrival times, ad hoc changing orders of incoming railcars, the increasing number of rolling stock, sparse capacities, and financial constraints complicate the process and offer large potential for optimization.

In the above context, we provide a description of a quite general class of problems called SORTING OF ROLLING STOCK – in the following SRS for short – that cover a broad range of special applications. In general, such problems consist of three processes: arrival, parking, and departure. At the beginning an ordered *input sequence* of *units* of rolling stock (railcars, trams, complete trains, ...) arrives at the rail yard. Then the parking process starts and the units enter the tracks of the rail yard. Here, incoming units have to be parked in such a way that at departure time the parked units can leave the rail yard in a structured *output sequence*. Note that the output sequence may contain information for several outbound trains.

The difficulty of SRS depends on the structural differences of the input sequence and the requested output sequence, as well as on the structure and flexibility of the rail yard.

**Figure 1.3.** One of the rail yards at the site of our practical partner: BASF, The Chemical Company, Ludwigshafen

**Structure of Output Sequence** As usual the incoming units are classified by a particular distinctive criterion, e. g., their destination or their construction type. As common in practice, we say that units satisfying the same criterion form a *group*.

We distinguish the following different structures of output sequences. Suppose, all positions of the output sequence are labeled, e. g., with letters, and all assigned units departing at positions with identical label are members of the same group; and, vice versa, all members

from a group are assigned to positions with identical labels. In particular, the number $g$ of different groups is the same as the number of different labels. The labels of the positions of the output sequence form certain *patterns*. For example, consider the input sequence $(2,3,2,1)$ of four units: the first and third incoming unit belong to group 2, the second to group 3, and the last incoming unit to group 1. Assume that the pattern $(u,v,w,v)$ was requested for the output sequence. For output sequences with a **free $g$-pattern**, there is no fixed assignment between the $g$ groups and the labels. The desired **free 3-pattern** $(u,v,w,v)$ allows two configurations of the output sequence, namely $(1,2,3,2)$ and $(3,2,1,2)$. On the contrary, if there is a fixed assignment between the $g$ groups and the labels – units departing at some position of the output sequence have to be members of a pre-defined group – we speak of output sequences with **ordered $g$-pattern**. If the assignment in the above example were $u \mapsto 3$, $v \mapsto 2$, and $w \mapsto 1$, then the requested configuration of the output sequence would read $(3,2,1,2)$.

We say that the output sequence has a *block* pattern, if the positions of the output sequence are labeled in a blockwise manner, that is, if the output sequence contains no subsequence of positions labeled $(u,v,u)$ for distinct labels $u \neq v$. An output sequence with a block pattern has the structure **free $g$-blocks** if there is no pre-defined assignment between the $g$ groups and the labels, and **ordered $g$-blocks** otherwise. Thus, if the structure **free $g$-blocks** is required, there are $g!$ feasible configurations (block patterns) of the output sequence according to $g!$ possible orders of the groups at departure; for the input sequence $(2,3,2,1)$ they read $(1,2,2,3)$, $(1,3,2,2)$, $(2,2,1,3)$, $(2,2,3,1)$, $(3,1,2,2)$, and $(3,2,2,1)$. In the **ordered $g$-blocks** case, we only get one feasible configuration of the output sequence. For the above input sequence, that is $(1,2,2,3)$ for the block pattern $(u,v,v,w)$ with the assignment $u \mapsto 1$, $v \mapsto 2$, and $w \mapsto 3$.

In most cases, if it is required that the output sequence has one of the above-mentioned structures – **free $g$-pattern**, **ordered $g$-pattern**, **free $g$-blocks**, or **ordered $g$-blocks** – then SRS corresponds to forming one outbound train with the respective structure on one output track, see Figure 1.4. On the contrary, we say the output sequence has the structure **$o$-ordered $g$-blocks** (or **$o$-ordered $g$-pattern**) if it enables an assembly of $o$ outbound trains on $o$ parallel output tracks – without additional rearrangements – such that each outbound train has the desired structure **ordered blocks** (**ordered pattern**), see Chapter 3 for a more detailed description.

Note that the departing order of units within a group is not fixed and offers potential for optimization.

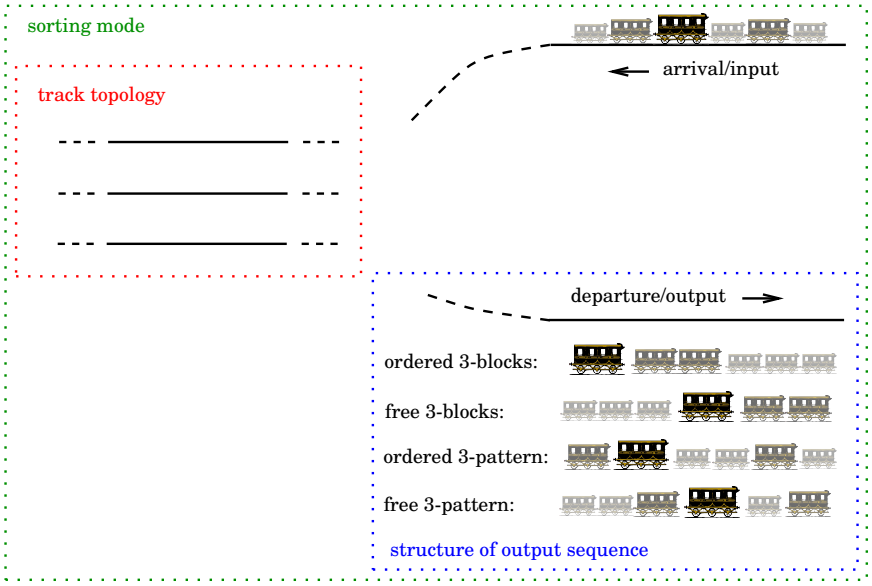


**Figure 1.4.** Sorting of Rolling Stock in rail yards

Of course, the choice of tracks is affected by several other parameters of the rail yard that specify either the track topology or the required sorting mode.

**Track Topology** There are various track topologies depending on the design and the length of the sorting tracks in a rail yard. In the following, if we speak of *tracks* we refer only to the sorting tracks in the rail yard.

*Design.* If the tracks may be accessed only from one side – that is, entrance and exit are on the same side of the tracks, such that the other end is a dead-lock – we speak of **stacks**. Note that any two units parked on the same stack will change their order from arrival to departure if they are not additionally rearranged or shunted. Under the same assumption, any two units preserve their order from arrival to departure when placed on a queue (**queues**), which is a one way track where the units arrive at one end and leave at the opposite side. In the case denoted as **stacks/queues**, one may freely decide whether a track is used as a queue or stack track. In the above three cases, both

the entrance and exit are only on one – possibly differing – side of the track, which is known in the literature as *siso* (single in single out). In the following track designs, units may arrive at or depart from both sides of the tracks: **sido** (single in double out), i. e., entrance is on one side, exit is on both sides; **diso** (double in single out), i. e., entrance is on both sides, exit is on one side; **dido** (double in double out), i. e., entrance and exit are on both sides. With respect to the number of tracks $t$ in the rail yard we denote the above-mentioned track designs by $t$-**stacks**, $t$-**queues**, and so on.

*Length.* Of course, in real rail yards, tracks are bounded in length. In the case $b$-**bounded**, at most $b$ units may be placed on each track. Though **unbounded** track lengths are seemingly a rather theoretical issue, they may well be reasonable from a practical point of view. In general, it is much harder and much more time-consuming to determine optimal schedules that comply with the real track lengths. Additionally, in practice, solutions that are optimal with respect to unbounded tracks seem to be easily transformable into efficient schedules for bounded tracks. For example, there are two well-known approaches for handling "*overfilled*" tracks during actual operations. Immediately after a track gets filled to capacity, one may either empty it using an additional buffer – that is, tracks beyond the rail yard – or redirect units initially planned to go on the filled track to another track in the rail yard.

**Sorting Mode** The technical and organizational infrastructure of the yard leads to different constraints on the feasible movements of units. We mainly capture these differences in the distinctive feasible movements available in a sorting mode: a unit may move or may be moved from the input(-track) to a track (*i-t-move*), from a track to another or the same track (*t-t-move* or *shunting-move*), from a track to the output(-track) (*t-o-move*) or directly from the input(-track) to the output(-track) (*i-o-move*). If the structures of the input sequence and the output sequence differ, at least some i-t-moves and t-o-moves are necessary.

*Shunting.* Now, let us consider special cases for which i-o-moves and/or different kinds of shunting-moves are permitted or prohibited. For instance, in the **no shunting** case, no shunting-moves and no i-o-moves are permitted. Otherwise, depending on the actual infrastructure of the rail yard, there are various constraints on the type of

feasible shunting-moves. If the rail yard features a so called *hump* for splitting trains, we speak of a *hump yard*. At the site of our project partner BASF in Ludwigshafen, the sorting and shunting operations are mainly performed using a large and expensive hump yard facility. Sorting or shunting over a hump is a common strategy for rearranging trains of units *without* own power units. Upon arrival such units are pushed over the hump before rolling down one by one onto either appropriately chosen tracks (i-t-moves) or the output-track (i-o-moves). As a consequence, instead of many time-consuming pushing/pulling operations of units by locomotives on tracks, only one pushing operation of the complete input sequence at arrival is needed. In the same convenient manner one may use the hump for t-t-moves and t-o-moves. For example, at BASF t-t-moves and t-o-moves are performed in the following fashion. At each *humping step* all units placed on one track are pulled back over the hump. Then, these units are again pushed over the hump either to the output-track (t-o-move) or to other tracks (t-t-move). In case of $h$-**hump-shunting**, we allow at most $h$ such humping steps. The only difference between **no shunting** and 0-**hump-shunting** is that i-o-moves are infeasible for **no shunting** but feasible for 0-**hump-shunting**. If shunting-moves are allowed, it is necessary to describe the performance of the required shunting-moves within the schedules to compute.

*Timing.* We distinguish cases in which arrival (i-o-move or i-t-move) and departure (t-o-move or i-o-move) appear completely separated or mixed on the time line. For example, at night depots it is quite common that the first outgoing unit departs in the morning, long after the parking process is completely finished at night. In this case – no i-o-moves and first t-o-move after last i-t-move – we say that arrival and departure are **sequential**. Otherwise, if we allow that departure and arrival are **concurrent** – i-o-moves are allowed or i-t-moves and t-o-moves do not need to be performed consecutively – we may freely choose the departure time of any track-leaving unit. However, if we want to minimize the number of tracks used, then a unit or group should obviously leave a track as soon as possible, thus reducing the chance of blockades of departures of other units or groups. The input information for the **sequential** as well as the **concurrent** versions is the input sequence, i. e., one only knows in which order the units arrive. Contrary to these **sequence** versions, in the more general **time windows** case, more input information has to be taken into account. Here, the arrival time and departure time for each unit are exactly

fixed in advance. We assume that units with identical departure time belong to the same group.

*Splitting.* Finally, the sorting mode is influenced by the way units may depart from tracks, which is closely related to the type of the units. Suppose our problem consists in sorting units with their own power units such as trams. Such units are able to leave the shunting yard without the help of other devices like locomotives. Thus, it is possible to split the units of one group arbitrarily over the tracks (**split**). However, this splitting might not be reasonable if we want to sort railcars without their own power units into **blocks**, since then one or more locomotives would have to collect the units of one group from several tracks. It would be much less time-consuming for the locomotive to pick all units of one group as a block from a single track. As a consequence, we also consider $s$**-split** versions in which the units of one group may only be split up over at most $s+1$ tracks with $s \geq 0$. The particularly restrictive splitting condition **chain-split** is reasonable only for **sequential** and **no shunting**. In such cases, units may be distributed over all tracks in such a way that collecting the units track by track – that is, all units placed on a track depart completely before all units of the next track depart etc. – leads to the required output sequence.

| **track topology** | | **sorting mode** | | | **structure of** | |
|---|---|---|---|---|---|---|
| design | length | shunting | timing | splitting | **output sequence** | |
| $t$-**st**acks | **u**n**b**ounded | **n**o **sh**unting | **se**quential | $s$-**sp**lit | **fr**ee | $g$-**bl**ocks |
| $t$-**qu**eues | $b$-**b**oun**d**ed | $h$-**h**ump-**sh**unting | **co**ncurrent | **sp**lit | **or**dered | $g$-**pa**ttern |
| $t$-**s**tacks/**q**ueues | | | **t**ime **w**indows | **c**hain-**sp**lit | $o$-**or**dered | |
| $t$-**s**i**d**o | | | | | | |
| $t$-**di**s**o** | | | | | | |
| $t$-**di****d**o | | | | | | |

**Table 1.1.** Parameters for SRS ($b \geq 1$, $h \geq 0$, $s \geq 0$, $g \geq 1$)

Similar to the notation for scheduling problems, we propose an $\alpha|\beta|\gamma =: \mathcal{V}$ notation for the description of the many different *versions* $\mathcal{V}$ of SRS which result from the specification of the various parameters. Here, $\alpha$ specifies the track topology, $\beta$ the sorting mode, and $\gamma$ the structure of the output sequence. A complete detailed list using the abbreviations defined in Table 1.1 reads as follows:

$\alpha \in \{\{t\textbf{-st}, t\textbf{-qu}, t\textbf{-sq}, t\textbf{-sd}, t\textbf{-ds}, t\textbf{-dd}, \cdot\} \times \{\textbf{ub}, b\textbf{-bd}, \cdot\}\}$,
$\beta \in \{\{\textbf{nsh}, h\textbf{-hsh}, \cdot\} \times \{\textbf{se}, \textbf{co}, \textbf{tw}, \cdot\} \times \{s\textbf{-sp}, \textbf{sp}, \textbf{csp}, \cdot\}\}$, and
$\gamma \in \{\{\textbf{fr}, \textbf{or}, o\textbf{-or}, \cdot\} \times \{g\textbf{-bl}, g\textbf{-pa}, \cdot\}\}$.

Note that an $\alpha|\beta|\gamma$ notation containing dots refers to all possible versions with requirements defined by the parameters differing from a dot. For example, a $\cdot$**,**$\cdot|$**nsh,se,**$\cdot|\cdot$**,**$\cdot$ version is any of all feasible **no shunting** and **sequential** versions.

**Objective** Of course, in practice, the goal is to perform the whole process of classifying inbound trains and of forming the outbound trains in a fast and cost-efficient way. As most practitioners will confirm, the following rule of thumb generally holds: the less tracks used and the less shunting performed, the faster the sorting and the lower the operational costs.

The objective to use as few tracks as possible is particularly reasonable if no shunting moves are allowed. In $t$**-minimizing** versions an optimal solution is a *schedule* – that is, an assignment of units to tracks – occupying only a minimum number of tracks for the required sorting.

For rail yards featuring a hump the size of the yard and the volume of traffic determine whether it is more efficient to perform the sorting with a minimum number of tracks for a given upper bound on the number of humping steps or rather with as few humping steps as possible for a given number of tracks. We refer to the latter objective as $h$**-minimizing**.

From a practical point of view, it might even be interesting to consider $b$**-minimizing** versions in order to compute schedules that utilize the $b$**-bounded** tracks symmetrically.

In our short descriptions $\alpha|\beta|\gamma$ of the versions we highlight the objective by an underline of the respective value which is to be minimized. For example, $\underline{t}$**-st,**$b$**-bd**$|h$**-hsh,se,sp**$|$**or,**$g$**-bl** is a $t$**-minimizing** version. If there is no such underline, then we address the corresponding decision problem that seeks for the answer of whether or not there is a feasible schedule regarding the specific parameters.

Note that a few combinations of the parameters listed in Table 1.1 do not make sense. For example, the combination of **free** and **time windows** is inconsistent, since a priori known departure times of the

units obviously result in a particular departure order, as in the **ordered** case. Nevertheless, there exist at least 100 different applicable versions of SRS.

## Outline of Thesis

The next chapter provides relevant notions and basic definitions in order to establish the theoretical basis for this thesis. In particular, we remind some notation for integer sequences and we list required graph classes and corresponding graph theoretical results.

In Chapter 3, we introduce mathematical formulations for the considered SRS versions. It is shown that many versions are closely related to particular graph coloring, scheduling, and bin packing problems. At the end, we list obvious relations between SRS versions.

Chapter 4 is devoted to classify the computational complexity of various SRS versions. Here, we survey known results. Additionally, we provide new complexity results; particularly, for versions that relate to the real-world optimization problem researched in our project together with BASF. Algorithms for relevant polynomially solvable versions are presented, and for a few $\mathcal{NP}$-hard versions approximability results are given.

Chapter 5 is concerned with the problem of finding a minimum vertex coloring of polygon-circle graphs, due to the fact that a few versions of SRS are equivalent to this problem. We present heuristical as well as exact solution approaches for determining optimal colorings of polygon-circle graphs. The proposed exact approaches consist in LP based and Lagrangian branch-and-bound procedures that are based on two different binary programming models: a new min cost flow formulation with side constraints and a classical assignment formulation.

In Chapter 6, we discuss our computational experience with solving SRS versions for real input data. We compare the computational times of different solution approaches for determining optimal schedules. For the required rearrangements at BASF we evaluate the quality of the computed schedules from a theoretical as well as from a practical point of view.

At the end, we give a brief introduction to competitive analysis for online algorithms in Chapter 7, and we classify the competitiveness of some online SRS versions.

CHAPTER 2

# Preliminaries

In this chapter, we list the notions and definitions of the fields Graph Theory, Complexity Theory, and Discrete Optimization that are relevant for this thesis. There is a broad range of literature on Graph Theory, see BRANDSTÄDT ET AL. (1999) , GOLUMBIC (2004) and GROSS & YELLEN (2005); on Discrete Optimization, see PAPADIMITRIOU & STEIGLITZ (1998), KORTE & VYGEN (2008), and SCHRIJVER (2003); as well as on Complexity Theory, see GAREY & JOHNSON (1979), PAPADIMITRIOU (1994), AUSIELLO ET AL. (1999), and VAZIRANI (2001). Nevertheless, we summarize basic concepts in order to make the thesis more self-contained for the reader's convenience.

## 2.1 Problems, Algorithms, and their Complexity

A *mathematical problem* $\mathfrak{P}$ is described by all instances $\mathfrak{I} \in \mathfrak{P}$. An *algorithm* can be seen as a finite set of instructions that perform operations on certain data of the instances. We say an algorithm *solves* $\mathfrak{P}$ if it finds a *feasible answer* for every $\mathfrak{I} \in \mathfrak{P}$. Among mathematical problems, we distinguish the class of *decision problems* whose instances exactly allow one of the feasible answers "*yes*" or "*no*" and the class of *optimization problems*. In optimization problems which are subclassified in *minimization* and *maximization problems*, the instances consist of a *set of solutions* $X$ and an *objective function* $f : X \to \mathbb{R}$. As an example of a minimization problem, let us consider a LINEAR PROGRAM where $X$ is a rational *polyhedron* – that is, $X = \{x \in \mathbb{R}^n | Ax \geq e\}$ for $A \in \mathbb{Q}^{m \times n}$, $e \in \mathbb{Q}^m$ – and where a linear objective function $f(x) = c^T x$ is

to be minimized. For an instance of such a problem, the feasible answer is either the *optimal value* $f(x^*) = \min_{x \in X} f(x)$ for an *optimal solution* $x^* \in X$, or "*infeasible*" if $X = \emptyset$, or "*unbounded*" if for any $\mathfrak{r} \in \mathbb{R}$ there is an $x \in X$ with $c^T x < \mathfrak{r}$.

Roughly speaking, the *computational complexity* of a problem is defined by the best – that is, "*fastest*" – algorithm solving this problem. If an algorithm is run on a computer, its actual running time depends on the implementation as well as the hardware. As a consequence, in theoretical terms, the *computational time* of an algorithm refers to the growth of the number of *elementary operations* performed with respect to increasing *size* of the instances. For example, elementary operations perform the addition, substraction, multiplication, division, or comparison of two values. For strict and extensive definitions of these operations we refer to PAPADIMITRIOU (1994). The size of a particular instance corresponds to the number of bits required to store all the data that defines the instance. This number depends on the encoding scheme. For example, the size of an integer $i$ is $1 + \lceil \log_2(|i| + 1) \rceil$ in case of the common binary encoding. We say the *time complexity* of an algorithm for solving a problem $\mathfrak{P}$ is $\mathcal{O}(f(|\mathfrak{I}|))$ if the number of elementary operations necessary for solving any $\mathfrak{I} \in \mathfrak{P}$ of size $|\mathfrak{I}|$ is bounded from above by $c \cdot f(|\mathfrak{I}|)$ for sufficiently large $|\mathfrak{I}|$, where $f : \mathbb{N} \to \mathbb{R}$ and $c$ is some constant.

An algorithm solves a problem *in polynomial time* if its time complexity is $\mathcal{O}(n^c)$ for some constant $c$. The class $\mathcal{P}$ contains all problems that can be solved in polynomial time. It is common parlance to say a problem $\mathfrak{P} \in \mathcal{P}$ is "*easily solvable*", since most problems in $\mathcal{P}$ can be solved on a computer quite fast, at least for moderate instance size and an efficient implementation of the applied algorithm.

For a broad range of mathematical decision problems – contained in a class which we define in the following – it is not known whether or not they are polynomially solvable.

A binary string $\mathfrak{B}$ of length polynomially bounded in the size of the instance $\mathfrak{I}$ is called *yes certificate for* $\mathfrak{I}$ if it ascertains that $\mathfrak{I}$ is an instance with the feasible answer "*yes*". A decision problem is in the class $\mathcal{NP}$ if there is a *certificate-checking algorithm* which for given $\mathfrak{I}$ and $\mathfrak{B}$ answers in polynomial time – in the size of the instance $\mathfrak{I}$ – whether $\mathfrak{B}$ is a yes certificate for $\mathfrak{I}$. While it is easily seen that $\mathcal{P} \subseteq \mathcal{NP}$, it is unknown whether there exists a decision problem in $\mathcal{NP}$ which is definitely not solvable in polynomial time. Proving $\mathcal{P} \neq \mathcal{NP}$ or $\mathcal{P} = \mathcal{NP}$ would answer the probably most popular open question in computer

science and mathematical optimization.

We say that a decision problem $\mathfrak{P}_1$ is *polynomially reducible* to $\mathfrak{P}_2$ if there is an algorithm which for every instance $\mathcal{I}_1$ of $\mathfrak{P}_1$ constructs an instance $\mathcal{I}_2$ of $\mathfrak{P}_2$ in polynomial time such that the answer for $\mathcal{I}_2$ is "*yes*" if and only if the answer for $\mathcal{I}_1$ is "*yes*". A problem $\mathfrak{P}$ is said to be $\mathcal{NP}$-hard if every problem in $\mathcal{NP}$ is polynomially reducible to $\mathfrak{P}$. If additionally $\mathfrak{P} \in \mathcal{NP}$, then $\mathfrak{P}$ is called $\mathcal{NP}$-complete. To prove the $\mathcal{NP}$-completeness of a problem $\mathfrak{P} \in \mathcal{NP}$, it suffices to show that a known $\mathcal{NP}$-complete problem is polynomially reducible to $\mathfrak{P}$. Many interesting problems in $\mathcal{NP}$ are shown to be $\mathcal{NP}$-complete. The existence of a polynomial time algorithm for an $\mathcal{NP}$-hard problem would prove $\mathcal{P} = \mathcal{NP}$. However, under the widely believed assumption $\mathcal{P} \neq \mathcal{NP}$, there exist no polynomial time algorithms for $\mathcal{NP}$-hard problems.

Above concepts describing the hardness of finding solutions were actually introduced for decision problems. However, with a slight lack of accuracy, the same terms are commonly applied to optimization problems. For a minimization problem consider the corresponding decision problem of asking if the optimal value $f(x^*)$ is less than or equal to a constant $c$. This decision problem is no harder than the minimization problem, assuming that the objective value can be evaluated in polynomial time. As a consequence, any proven result about the hardness of the decision problem carries over to the minimization problem. A minimization problem is said to be $\mathcal{NP}$-hard if the corresponding decision problem is $\mathcal{NP}$-hard. The $\mathcal{NP}$-hardness of an optimization problem $\mathfrak{P}$ indicates that an algorithm for solving $\mathfrak{P}$ run on a computer might fail to compute optimal solutions for instances of large size in adequate time. If this is the case, a common and reasonable strategy is to develop and implement fast polynomial time algorithms that determine solutions provably close to optimality.

Consider a minimization problem $\mathfrak{P}$ and a polynomial time algorithm $\mathfrak{A}$ for solving $\mathfrak{P}$. For an instance $\mathcal{I} \in \mathfrak{P}$, we denote the objective values of the solution determined by $\mathfrak{A}$ and of an optimal solution by $z_{\mathfrak{A}}(\mathcal{I})$ and $z^*(\mathcal{I})$, respectively. If $z_{\mathfrak{A}}(\mathcal{I}) \leq \rho(|\mathcal{I}|) \cdot z^*(\mathcal{I})$ for any $\mathcal{I} \in \mathfrak{P}$, then $\mathfrak{A}$ is called $\rho(|\mathcal{I}|)$-*approximation algorithm* for $\mathfrak{P}$, where $\rho : \mathbb{N} \to \mathbb{R}$ is the *performance guarantee*. Problem $\mathfrak{P}$ is said to be $\rho(|\mathcal{I}|)$-*approximable* if there is a $\rho(|\mathcal{I}|)$-*approximation algorithm* for $\mathfrak{P}$. If $\rho$ is a constant function, then $\mathfrak{A}$ is called a *constant factor approximation*. We say $\mathfrak{P}$ is $+c$-*approximable* if $c$ is a constant and if there is a polynomial time algorithm $\mathfrak{A}$ for $\mathfrak{P}$ such that $z_{\mathfrak{A}}(\mathcal{I}) \leq z^*(\mathcal{I}) + c$ for any $\mathcal{I} \in \mathfrak{P}$.

## 2.2 Combinatorial Optimization

Linear Combinatorial Optimization aims at algorithmically solving mathematical optimization problems over discrete structures by combining techniques from Linear Programming and Combinatorics. In this field, the developed solution methods apply to many real-life optimization problems – for example, in management, production, and logistics. Typical applications are concerned with an efficient allocation of limited resources to meet desired objectives when "*yes*" or "*no*" decisions are involved or when the values of some or all of the variables are restricted to be integral.

### 2.2.1 Problems

Many of the well-known Combinatorial Optimization problems can be formulated as MIXED INTEGER (LINEAR) PROGRAMS – henceforth referred to as MIP for short – which are problems of the form

$$\min \left\{ c^T x \;\middle|\; Ax \geq e,\ x \in X \right\} \tag{2.1}$$

with $c \in \mathbb{Q}^n$, $A \in \mathbb{Q}^{m \times n}$, $e \in \mathbb{Q}^m$, and $X = \mathbb{Z}_+^l \times \mathbb{R}_+^{n-l}$ for $1 \leq l < n$. If $X = \{0,1\}^l \times [0,1]^{n-l}$, $X = \mathbb{Z}_+^n$, or $X = \{0,1\}^n$ in above definition, we speak of a MIXED BINARY (LINEAR) PROGRAM (MBP), an INTEGER (LINEAR) PROGRAM (IP), or a BINARY (LINEAR) PROGRAM (BP), respectively. Of course, an MBP is a particular MIP, and each BP belongs to the class of IPs.

In the following, we list known $\mathcal{NP}$-hard Combinatorial Optimization problems that are relevant for this thesis. Note that they all relate to graph coloring and they all can be formulated as BPs.

#### Minimum Vertex Coloring

Before stating the MINIMUM VERTEX COLORING problem, we introduce some fundamentals of Graph Theory. Let $V$ be a finite set of *vertices* (*nodes*) and let $\mathcal{P}^2(V)$ denote the set of all 2-element subsets of $V$. An *undirected graph* $G = (V, E)$ is defined by $V$ and the set $E \subseteq \mathcal{P}^2(V)$ of *edges*. By this definition, undirected graphs do not contain parallel edges and self-loops. A *directed graph* $G = (V, E)$ is defined by $V$ and the set $E \subseteq V \times V$ of *directed edges* or *arcs*. If nothing else is stated, graphs are understood to be undirected.

Two vertices $u$ and $v$ are called *adjacent* if and only if $\{u,v\} \in E$. $N_G(v) = \{u \in V \,|\{u,v\} \in E\}$ denotes the *neighborhood* of vertex $v$ in graph $G = (V,E)$. A graph $G[V'] = (V', E(V'))$ is an *induced subgraph* of $G = (V,E)$ if $V' \subseteq V$ and $E(V') = \{\{u,v\} \,|\, u, v \in V', \{u,v\} \in E\}$. The *complement graph* of graph $G = (V,E)$ is denoted by $coG = (V, coE)$, where $coE = \{\{u,v\}|\{u,v\} \notin E\}$. We say that two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a bijective function $f$ from $V_1$ onto $V_2$ such that $\{u,v\} \in E_1$ if and only if $\{f(u), f(v)\} \in E_2$.

The number of elements in a set $M$ is called *cardinality* of $M$, and it is denoted by $|M|$. A subset $C \subseteq V$ is called *clique* in $G$ if and only if $\{u,v\} \in E$ for all $u,v \in C$ with $u \neq v$. The *clique number* $\omega(G)$ of graph $G$ is defined by $\omega(G) := \max\{|C| : C \subseteq V$ and $C$ is a clique in $G\}$. The subset $I \subseteq V$ is said to be an *independent set* in $G$ if and only if $\{u,v\} \notin E$ for all $u,v \in I$. The *independence number* $\alpha(G)$ of $G$ is defined by $\alpha(G) := \max\{|I| : I \subseteq V$ and $I$ is an independent set in $G\}$. A ($k$-)*coloring* is a mapping $f : V \to \{1,\ldots,k\}$ such that no two adjacent vertices have the same color. The *chromatic number* of $G$ – denoted by $\chi(G)$ – is the minimal $k$ for which $G$ admits a $k$-coloring.

A directed graph $G = (V,E)$ is said to be *transitive* if $(u,v) \in E$ and $(v,w) \in E$ implies $(u,w) \in E$. A graph $G = (V,E)$ is a *comparability graph* if it has a *transitive orientation*, that is, there is an assignment of directions to the edges of $G$ such that the resulting directed graph is transitive.

The problem MINIMUM (VERTEX) COLORING abbreviated as MVC is to find a coloring of a given graph $G$ with $\chi(G)$ colors. For a formulation of this problem as BP, see Section 5.4.1.

### Bin Packing with Conflicts

Let be given a set of items $V = \{1,\ldots,n\}$ with sizes $w_1,\ldots,w_n$ being rational numbers in the interval $[0,1]$ and a conflict graph $G = (V,E)$. The problem of BIN PACKING WITH CONFLICTS – BPC for short – is to store the items into a minimum number of bins of size one, such that two conflicting, i.e., adjacent, items are not stored into the same bin. In other words, the goal is to find a partition of $V$ into a minimum number of independent sets of $G$ where the total size of each independent set – obtained by summing up the sizes of all items contained in the independent set – is at most one. BPC generalizes both the classical (one-dimensional) BIN PACKING problem where $E = \emptyset$ and MVC where $w_i = 0$ for all $i = 1,\ldots,n$.

**Mutual Exclusion Scheduling**

As the name suggests, the following MUTUAL EXCLUSION SCHEDULING problem was originally defined in terms of scheduling: find an assignment (schedule) of $n$ given jobs with unit processing time to $b$ machines that yields the minimum *makespan* (the minimum completion time of the job finished last), when some of the jobs are not allowed to be processed at the same time. In the literature, this problem is commonly denoted as $b$-MES for short.

It is easy to see that $b$-MES is equivalent to the following graph coloring problem: find a feasible vertex coloring of a given graph $G$ with minimum number of colors such that at most $b$ vertices are colored with the same color. Note that a vertex of $G$ corresponds to a job and each color relates to a certain time slice $[i, i+1]$ with $i \in \mathbb{N}$.

For the rest of this thesis, $b$-MES refers to this formulation in terms of graph coloring. Of course, $b$-MES is a specialization of BPC where the vertices (items) all have size $1/b$.

### 2.2.2 Solution Methods

For many combinatorial optimization problems, it is not too hard to come up with an intuitive strategy for determining feasible solutions. A resulting detailed algorithm is said to be a *heuristical* method if it is easy to implement and if it produces solutions which are hopefully close to the optimal solution. If we can prove a performance guaranty of such a method, we speak of an *approximation*, see Section 2.1 for a formal definition. Let us give an example of a heuristical method for the $\mathcal{NP}$-hard MVC. Assume – for the rest of this thesis – that the colors that need to be assigned to the vertices of a graph are expressed by positive integers. For MVC, the algorithm denoted by FIRST FIT is a procedure which iteratively "*colors*" the vertices of a given graph – according to a pre-defined order – with the smallest color that ensures a feasible coloring. Note, FIRST FIT is easily adoptable to $b$-MES; we only need to prevent the assignment of a particular color if we already colored $b$ vertices with this color.

For complex real-world optimization problems, it is often not trivial to develop methods that rapidly compute good solutions. However, Combinatorial Optimization faces the ambitious challenge of determining optimal solutions in adequate time. Algorithms which output the optimal solution at the end are referred to as *exact* methods. For prob-

lems of huge dimension and complexity, many exact solution methods rely on the traditional divide-and-conquer strategy, that is, they deal with an adequate sequence of subproblems. Probably the most famous exact approach for solving IPs or MIPs is *branch-and-bound*. Let us consider such a problem of the form $\min_{x \in X} f(x)$. As the name suggests, a branch-and-bound method requires two tools, the branching procedure and the bounding procedure. The branching procedure returns – for a given subset $X_\mathfrak{s} \subseteq X$ of feasible solutions – two or more smaller sets whose union covers $X_\mathfrak{s}$. Its recursive application defines a tree structure – the search tree – whose nodes correspond to the subsets of $X$. The bounding procedure computes upper and lower bounds for the minimum value of a solution contained in subset $X_\mathfrak{s}$. Mostly, upper bounds are determined by heuristically computed solutions. The key idea of branch-and-bound is: if the easily computable lower bound on the minimum value of a solution contained in the current subset $X_\mathfrak{s}$ is greater than or equal to the value of the currently best known feasible solution in $X$, then $X_\mathfrak{s}$ surely does not contain a better solution and we can discard the corresponding branch of the search tree. Besides this so called *pruning*, we stop branching on a subset $X_\mathfrak{s}$ when $X_\mathfrak{s}$ is empty, when the best solution in $X_\mathfrak{s}$ – that is, $x^*_\mathfrak{s} \in X_\mathfrak{s}$ with $f(x^*_\mathfrak{s}) = \min_{x \in X_\mathfrak{s}} f(x)$ – can easily be obtained, or also when the upper bound for set $X_\mathfrak{s}$ matches the lower bound.

Obviously, branch-and-bound outputs an optimal solution – if one exists – at the end. However, how fast this happens critically depends on the effectiveness of the branching and bounding procedures that are implemented. Besides that, the following rule of thumb holds in most cases: the better the start solution – which can be determined by appropriate heuristics – the faster the computation. In practice, the computation is often interrupted after a chosen amount of time when no optimal solution is identified. At that point, the lower bound on the optimal value – which is given by the minimum of the lower bounds among all unpruned nodes – provides an *optimality gap* to the best known feasible solution.

Of course, branch-and-bound is a quite general framework for solving IPs, MIPs, and other combinatorial problems. In standard *LP based* branch-and-bound implementations, the bounding as well as the branching procedures exploit the *LP relaxation* of the current subproblem. The LP relaxation of an IP (MIP) results from relaxing integer variables $x_j \in \mathbb{Z}_+$ to $x_j \in \mathbb{R}_+$ and binary variables $x_j \in \{0,1\}$ to $x_j \in [0,1]$ within the given IP (MIP) formulation. In case that

the LP relaxations are strengthened by adding cutting-planes – e. g., see NEMHAUSER & WOLSEY (1988) – we speak of *branch-and-cut* procedures. A lower bound on the optimal value $z^*_{\mathfrak{s}}$ of the currently processed subproblem $\mathfrak{P}_{\mathfrak{s}}$ is obtained by solving the corresponding LP relaxation. In particular, the lower bound equals $z^{\text{LP}}_{\mathfrak{s}}$ where $z^{\text{LP}}_{\mathfrak{s}}$ is the value of the LP solution. An upper bound on $z^*_{\mathfrak{s}}$ may possibly be determined either by an LP solution that is also feasible for the IP (MIP) or by LP rounding techniques. On the branching side, we have to specify how to subdivide a subset $X_{\mathfrak{s}}$. The most common scheme is to choose an integer variable $x_j$ with a fractional value $\hat{x}_j$ in the solution of the current LP relaxation, and create one new subproblem by adding the inequality $x_j \leq \lfloor \hat{x}_j \rfloor$ and a second new subproblem with the additional constraint $x_j \geq \lceil \hat{x}_j \rceil$. The rule for subdividing a subset $X_{\mathfrak{s}}$ is said to be the *branching policy*. Besides that, whenever a subproblem has been processed, we need to decide which unpruned node (subproblem) should be examined in detail next. A corresponding strategy is referred to as *node selection policy*. In the literature, many different *branching* and *node selection policies* have been proposed. A profound discussion and evaluation of various schemes can be found in ACHTERBERG (2007).

Alternatively, the bounds can be computed by solving any other relaxation of the current subproblem; for example, we may exploit a *Lagrangian relaxation*. The basic idea of a Lagrangian relaxation is to relax some of the constraints and penalize the violation of these constraints within the objective function. For example, let us consider the following BP:

$$z^* = \min \left\{ c^T x \;\middle|\; Ax \geq e,\ Fx \geq q,\ x \in \{0,1\}^n \right\}. \tag{2.2}$$

Then, the problem

$$z^{\text{LR}}(\lambda) = \min \left\{ c^T x + \lambda^T (q - Fx) \;\middle|\; Ax \geq e,\ x \in \{0,1\}^n \right\} \tag{2.3}$$

is said to be a Lagrangian relaxation of (2.2). The components of $\lambda$ are called *Lagrangian multipliers*. Obviously, it holds $z^{\text{LR}}(\lambda) \leq z^*$ for fixed $\lambda \geq 0$. The optimal solution of the problem

$$z^{\text{LD}} = \max_{\lambda \geq 0} z^{\text{LR}}(\lambda), \tag{2.4}$$

which is called the *Lagrangian dual* of (2.3), corresponds to a vector $\lambda^*$ that produces the greatest lower bound on $z^*$. We also say that (2.3)

is the *inner problem* of the Lagrangian dual (2.4). It is easily seen that $z^{\mathrm{LP}} \leq z^{\mathrm{LD}} \leq z^*$, where $z^{\mathrm{LP}}$ is the optimal value of the LP relaxation of (2.2). Note, if $\{x \in [0,1]^n \mid Ax \geq e\}$ is integral, then $z^{\mathrm{LP}} = z^{\mathrm{LD}}$. Of course, the more structure of the original problem is described by the constraints of the inner problem, the better the lower bounds. The key for an efficient Lagrangian branch-and-bound is to compute good lower bounds, while the inner problem of the Lagrangian dual can still be solved rapidly.

## 2.3 Integer Sequences and Intervals

We denote an *integer sequence* by $S = (s_{i_1}, \ldots, s_{i_n})$ where $s$ is a surjective function mapping each *element* $i \in \{i_1, \ldots, i_n \mid i_1 < i_2 < \cdots < i_n\}$ to an *integer* $s_i \in \mathcal{G}^S = \{\mathfrak{g}_1, \ldots, \mathfrak{g}_g\}$ with $\mathfrak{g}_1 < \mathfrak{g}_2 < \cdots < \mathfrak{g}_g$. In particular, each integer $\mathfrak{g} \in \mathcal{G}^S$ occurs in $S$. In order to avoid confusion, we say that $S = (1,2,1)$ "*is an integer sequence of three elements containing two (different) integers*" rather than it "*is an integer sequence of three integers*". For the sake of clarity, $S_{n,g}$ denotes an integer sequence of $n$ elements which contains $g$ different integers. A *permutation* $\Pi = (\pi_1, \ldots, \pi_n)$ of the integers $1, \ldots, n$ corresponds to an integer sequence with $\mathcal{G}^S = \{1, \ldots, n\}$. Two integer sequences $S = (s_{i_1}, s_{i_2}, \ldots, s_{i_n})$ and $\bar{S} = (\bar{s}_{j_1}, \bar{s}_{j_2}, \ldots, \bar{s}_{j_n})$ are said to be *equal* if $s_{i_l} = \bar{s}_{j_l}$ for all $l = 1, \ldots, n$. Otherwise, the two sequences are called *different*. The concatenation $S \oplus \bar{S} = (\tilde{s}_1, \ldots, \tilde{s}_{n+\bar{n}})$ of two integer sequences $S = (s_{i_1}, s_{i_2}, \ldots, s_{i_n})$ and $\bar{S} = (\bar{s}_{j_1}, \bar{s}_{j_2}, \ldots, \bar{s}_{j_{\bar{n}}})$ is a binary operation defined by $\tilde{s}_k := s_{i_k}$ for $k = 1, \ldots, n$ and $\tilde{s}_k := \bar{s}_{j_{k-n}}$ for $k = n+1, \ldots, n+\bar{n}$. The *element reversing* sequence of $S = (s_{i_1}, \ldots, s_{i_n})$ is given by $S^{\leftrightarrow} = (\bar{s}_{i_1}, \ldots, \bar{s}_{i_n})$ with $\bar{s}_{i_j} = s_{i_{n-j+1}}$ for $j = 1, \ldots, n$. For example, $S^{\leftrightarrow}$ equals $(4,2,1,3)$ for $S = (3,1,2,4)$. The *integer reversing* sequence of $S$ is defined by $S^{\updownarrow} = (\bar{s}_{i_1}, \ldots, \bar{s}_{i_n})$ with $\bar{s}_{i_j} = \mathfrak{g}_{g-k+1}$ if $s_{i_j} = \mathfrak{g}_k$. For example, $S^{\updownarrow}$ equals $(3,4,3,2,1)$ for $S = (2,1,2,3,4)$.

Moreover, a *subsequence of* $S = (s_{i_1}, \ldots, s_{i_n})$ is an integer sequence $(s_{i_{j_1}}, s_{i_{j_2}}, \ldots, s_{i_{j_m}})$ such that $1 \leq j_k < j_l \leq n$ for $1 \leq k < l \leq m$. We say that an integer sequence $S$ *contains a subsequence* $(u,v)$, if there exists a subsequence $(s_i, s_j)$ of $S$ with $s_i = u$, $s_j = v$. For instance, $S = (3,1,2)$ contains a subsequence $(3,2)$. By *removing* a subsequence $S_1 = (s_{j_1}, \ldots, s_{j_m})$ from $S = (s_{i_1}, \ldots, s_{i_n})$ we get the remaining subsequence $S \backslash S_1 = (s_{l_1}, \ldots, s_{l_{n-m}})$ of $S$ with $l_k \in \{i_1, \ldots, i_n\} \backslash \{j_1, \ldots, j_m\}$ for $k = 1, \ldots, n-m$. The union $S_1 \cup S_2$ of two subsequences

$S_1 = (s_{j_1}, \ldots, s_{j_m})$ and $S_2 = (s_{l_1}, \ldots, s_{l_{\tilde{m}}})$ of an integer sequence $S$ equals $(s_{k_1}, s_{k_2}, \ldots, s_{k_{\tilde{m}}})$ with $k_t \in \{j_1, \ldots, j_m, l_1, \ldots, l_{\tilde{m}}\} =: J$, $\tilde{m} := |J|$, and $k_1 < k_2 < \cdots < k_{\tilde{m}}$. Of course, $S_1 \cup S_2$ is also a subsequence of $S$.

An integer sequence $S = (s_{i_1}, \ldots, s_{i_n})$ is called *increasing* if $s_{i_j} < s_{i_k}$ for $i_j < i_k$. It is said to be *decreasing* if $s_{i_j} > s_{i_k}$ for $i_j < i_k$, and we say it is *monotone* if it is either increasing or decreasing. A sequence $S = (s_{i_1}, \ldots, s_{i_n})$ is called *upper unimodal* if there is an $l \in \{1, \ldots, n\}$ such that $s_{i_1} < s_{i_2} < \cdots < s_{i_l}$ and $s_{i_l} > s_{i_{l+1}} > \cdots > s_{i_n}$. Finally, it is said to be *unimodec* if it is the union of an upper unimodal sequence $S_1 = (s_{j_1}, \ldots)$ and a decreasing sequence $S_2 = (s_{j_1}, \ldots)$, both starting with the same element $j_1$.

A set of subsequences $P_S$ of $S$ is called *partition of* $S$ if each element of $S$ belongs to exactly one subsequence in $P_S$. With respect to a certain property of subsequences – for example monotonicity – we call a subsequence of $S$ *feasible*, if it has the property, or *infeasible* otherwise. Then, a *minimum (feasible) partition of* $S$ partitions $S$ into a minimum number of feasible subsequences.

For any integer contained in integer sequence $S$, that is, for each $\mathfrak{g} \in \mathcal{G}^S$ let us define $\bar{n}_\mathfrak{g} := |\{i \,|\, s_i = \mathfrak{g}\}|$, $p^f_\mathfrak{g}(S) := \min\{\, i \,|\, s_i = \mathfrak{g}\}$, as well as $p^l_\mathfrak{g}(S) := \max\{\, i \,|\, s_i = \mathfrak{g}\}$. The interval $P_\mathfrak{g}(S) = \left[p^f_\mathfrak{g}(S), p^l_\mathfrak{g}(S)\right]$ is said to be the *period* of integer $\mathfrak{g} \in \mathcal{G}^S$. For a given sequence $S$, the set $\{P_1(S), \ldots, P_g(S)\}$ of all periods is denoted by $P(S)$.

We denote a set $\{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$ of $n$ intervals of the real line by $\mathcal{I}$. We say two intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ *intersect* if $\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset$, and more precisely, $\mathcal{I}_i$ and $\mathcal{I}_j$ *overlap* if $\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset$, $\mathcal{I}_i \not\subseteq \mathcal{I}_j$, and $\mathcal{I}_j \not\subseteq \mathcal{I}_i$, whereas $\mathcal{I}_i$ *contains* $\mathcal{I}_j$ if $\mathcal{I}_j \subset \mathcal{I}_i$. Finally, we use $\mathcal{I}_i \lessdot \mathcal{I}_j$ if and only if $\mathfrak{r}_i < \mathfrak{r}_j$ for all reals $\mathfrak{r}_i \in \mathcal{I}_i$ and $\mathfrak{r}_j \in \mathcal{I}_j$.

## 2.4 Relevant Graph Classes

In Subsection 2.2.1 we defined what a graph formally is. Now, we list the definitions of relevant graph classes as an overview: intersection graphs (defined by MARCZEWSKI (1945) (in French)), interval graphs (introduced by HAJÖS (1957)), overlap graphs (defined by FULKERSON & GROSS (1965)), perfect graphs (introduced by BERGE (1961)), permutation graphs (defined by PNUELI ET AL. (1971)), circle graphs (defined by EVEN & ITAI (1971)), point-interval graphs (defined by

CORNEIL & KAMULA (1987)), trapezoid graphs (introduced by DAGAN ET AL. (1988)), polygon-circle graphs (suggested by FELLOWS in 1988, see KOSTOCHKA & KRATOCHVIL (1997)), and interval-filament graphs (introduced by GAVRIL (2000)).

A graph

- is an *intersection graph of* $M$,
  if its vertex set corresponds to a set $M = \{M_1, \ldots, M_n\}$ of $n$ objects which also can be expressed as sets such that two vertices (objects) $M_i$ and $M_j$ are adjacent in $G_M$ if the intersection of $M_i$ and $M_j$ is nonempty, i. e., $M_i \cap M_j \neq \emptyset$,

- denoted by $G^I$ or $G^I(\mathcal{I})$ is an *interval graph*,
  if it is the intersection graph of a set of intervals $\mathcal{I}$,

- denoted by $G^{PI}$ or $G^{PI}(\Delta)$ is a *point-interval graph*,
  if it is the intersection graph of a set of triangles $\Delta = \{\, \Delta_i(\mathfrak{p}_1^i, \mathfrak{p}_2^i, \mathfrak{p}_3^i) \mid i = 1, \ldots, n \,\}$ whose corner points are on two parallel real lines $L_1$ and $L_2$ such that the two corner points $\mathfrak{p}_1^i$ and $\mathfrak{p}_2^i$ of the triangle $\Delta_i$ are on $L_1$ with $\mathfrak{p}_1^i < \mathfrak{p}_2^i$ and the third corner point $\mathfrak{p}_3^i$ of $\Delta_i$ is on line $L_2$, see Figure 4.2 on page 57,

- denoted by $G^T$ or $G^T(\mathcal{T})$ is a *trapezoid graph*,
  if it is the intersection graph of a set of trapezoids $\mathcal{T}$ whose corner points are on two parallel real lines $L_1$ and $L_2$,

- denoted by $G^C$ or $G^C(\mathcal{C})$ is a *circle graph*,
  if it is the intersection graph of a set of chords $\mathcal{C}$ of the same circle, see Figure 2.1,

- denoted by $G^{PC}$ or $G^{PC}(\mathcal{P})$ is a *polygon-circle graph*,
  if it is the intersection graph of a set of polygons $\mathcal{P}$ with corner points on the same circle, see Figure 2.3,

- denoted by $G^{IF}$ or $G^{IF}(\mathcal{F})$ is an *interval-filament graph*,
  if it is the intersection graph of a set of curves $\mathcal{F}$ such that each curve $f_i \in \mathcal{F}$ connects the two end points $a_i$ and $d_i$ of an interval $\mathcal{I}_i = [a_i, d_i]$ on the real line and two curves do not intersect if the corresponding intervals are disjoint, see Figure 2.4,

- denoted by $G^O$ or $G^O(\mathcal{I})$ is an *overlap graph*,
  if the vertices correspond to intervals of a set of intervals $\mathcal{I}$, and two vertices $\mathcal{I}_i$ and $\mathcal{I}_j$ are adjacent if the intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ overlap, see Figure 2.1,
- denoted by $G^P$ or $G^P(\Pi)$ is a *permutation graph*,
  if the vertices correspond to the elements of a permutation $\Pi$ such that two vertices $i$ and $j$ are adjacent if $i < j$ and $\pi_i > \pi_j$,
- is a *perfect graph*,
  if the clique number equals the chromatic number for all of its induced subgraphs.

We say $\mathcal{I}$ is the *interval representation* of $G^I(\mathcal{I})$, $\mathcal{P}$ is said to be the *polygon representation* of $G^{PC}(\mathcal{P})$, and so on.

In what follows, we will briefly show some relations between these graph classes. The class of intersection graphs is equivalent to the class of all (undirected) graphs, see MARCZEWSKI (1945).

It is also well-known that the class of overlap graphs is equivalent to the class of circle graphs, it can be obtained with the following projection method suggested by GAVRIL (1973). Note, for each overlap graph $G^O(\mathcal{I})$ there exists an overlap graph $G^O(\bar{\mathcal{I}})$ w. r. t. an interval representation $\bar{\mathcal{I}}$ whose intervals all have different start and end points which is isomorphic to $G^O(\mathcal{I})$. Similarly, for each circle graph $G^C(\mathcal{C})$ there exists a circle graph $G^C(\bar{\mathcal{C}})$ w. r. t. a set of chords $\bar{\mathcal{C}}$ whose (end) points are all distinct which is isomorphic to $G^O(\mathcal{C})$. For such a given set of chords $\bar{\mathcal{C}}$ between distinct points on a circle we label the points with real numbers such that the reals are clockwise – beginning from one point – increasing. We denote the point corresponding to real $\mathfrak{r}$ by $\mathfrak{p}(\mathfrak{r})$. It is easily seen, that two chords $\mathcal{C}_i = (\mathfrak{p}(a_i), \mathfrak{p}(d_i)) \in \bar{\mathcal{C}}$ and $\mathcal{C}_j = (\mathfrak{p}(a_j), \mathfrak{p}(d_j)) \in \bar{\mathcal{C}}$ are intersecting if and only if the two intervals $[a_i, d_i]$ and $[a_j, d_j]$ overlap, see Figure 2.1. On the contrary, from a given interval representation $\mathcal{I}$ we derive a set of chords denoted by $\mathcal{C}(\mathcal{I})$ such that the respective circle graph $G^C(\mathcal{C}(\mathcal{I}))$ and the overlap graph $G^O(\mathcal{I})$ are isomorphic. As a consequence, the class of overlap graphs and the class of circle graphs are equivalent.

For the definition of spider graphs and their relation to polygon-circle graphs, see next subsection. The containment relations among the graph classes are shown in Figure 2.2 on page 28. For further results on these graph classes, the reader is referred to BRANDSTÄDT ET AL. (1999) and GOLUMBIC (2004).

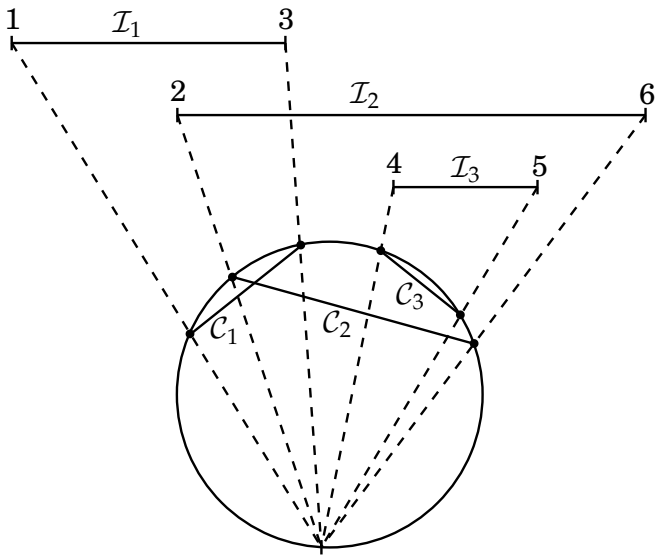


**Figure 2.1.** Relations between intervals and chords in a circle

### 2.4.1 Polygon-Circle Graphs

The projection method of GAVRIL (1973) described in the last section extends to polygon-circle graphs, see Figure 2.3 on page 29.

For a given set of polygons $\mathcal{P}$ with $n^{\mathcal{P}}$ polygons whose corner points are located on a circle line we assume that the points are labeled with non-negative real numbers such that the reals are clockwise – beginning from one point – increasing. By $\mathcal{P}_i(\mathfrak{R}_i)$ we denote the polygon with its $n_i^c$ corner points $\mathfrak{p}(\mathfrak{r}_k^i)$ corresponding to the reals $\mathfrak{r}_k^i \in \mathfrak{R}_i = \{\mathfrak{r}_1^i, \ldots, \mathfrak{r}_{n_i^c}^i\}$ with $\mathfrak{r}_{k_1}^i < \mathfrak{r}_{k_2}^i$ if $k_1 < k_2$. We denote the interval between $\mathfrak{r}_1^i$ and $\mathfrak{r}_{n_i^c}^i$ by $\mathcal{I}^{\mathcal{P}_i}$. The corner point $\mathfrak{p}(\mathfrak{r}_1^i)$ is called the *first corner point* and $\mathfrak{p}(\mathfrak{r}_{n_i^c}^i)$ the *last corner point* of $\mathcal{P}_i(\mathfrak{R}_i)$. The total number of corner points is given by $n^c(\mathcal{P}) := \sum_{i=1}^{n^{\mathcal{P}}} n_i^c$.

**Convention 2.1**
*For the rest of this thesis, we w.l.o.g. assume that a set of polygons denoted by $\mathcal{P}$ or by $\tilde{\mathcal{P}}$ only contains polygons with at least two corner points, and that the corner points of all polygons are pairwise disjoint. In $\mathcal{P}$ the polygons are increasingly numbered according to the order of their last corner points. In $\tilde{\mathcal{P}}$ the polygons are increasingly numbered according to the order of their first corner points.*

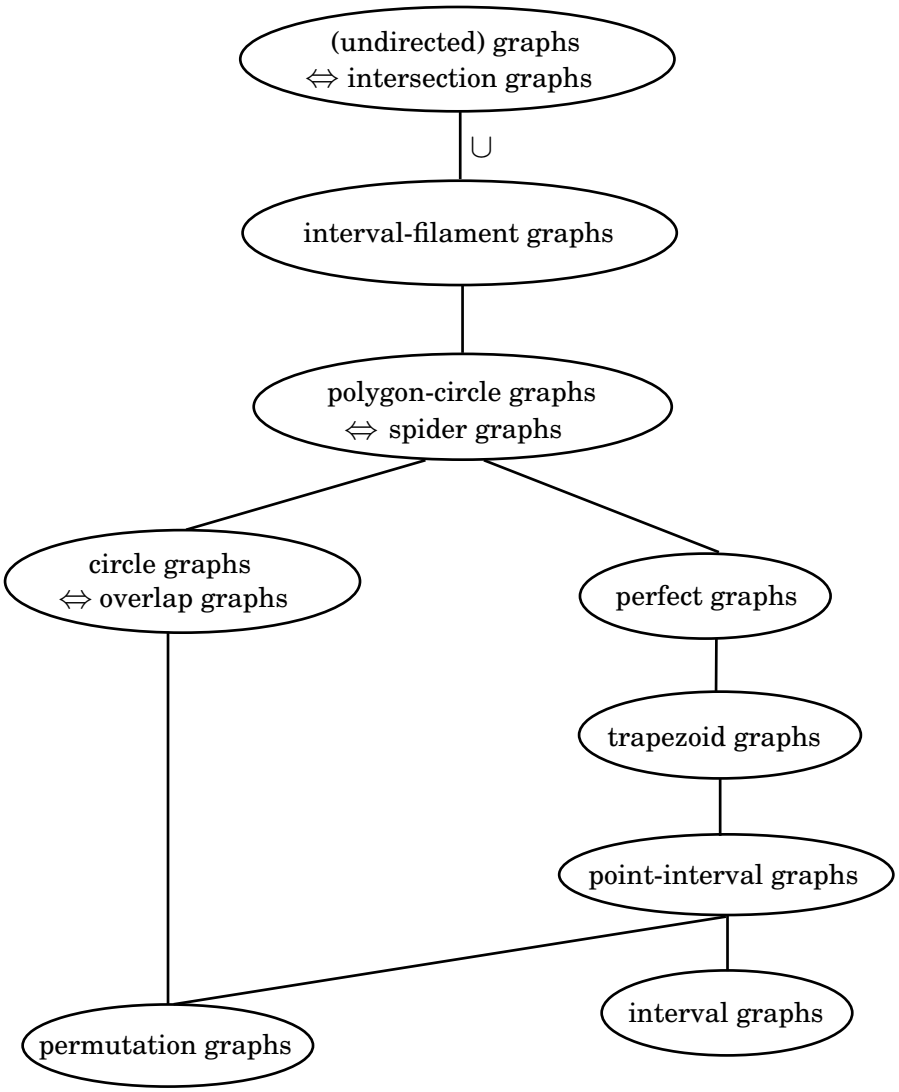


**Figure 2.2.** Containment relations of relevant graph classes

Note that for any set $\mathcal{P}'$ of polygons which violates one of the above conditions on $\mathcal{P}$, we can easily construct a polygon representation $\mathcal{P}$ such that the two corresponding polygon-circle graphs are isomorphic.

A graph is a *spider graph* – introduced by KOEBE (1992) – if the vertices correspond to sets of reals assigned to points on a circle in the above-mentioned way, such that two vertices $\mathfrak{R}_i$ and $\mathfrak{R}_j$ are adjacent if there exist reals $\mathfrak{r}^i_{k_1}, \mathfrak{r}^i_{k_2}$ of $\mathfrak{R}_i$ and reals $\mathfrak{r}^j_{l_1}, \mathfrak{r}^j_{l_2}$ of $\mathfrak{R}_j$ with

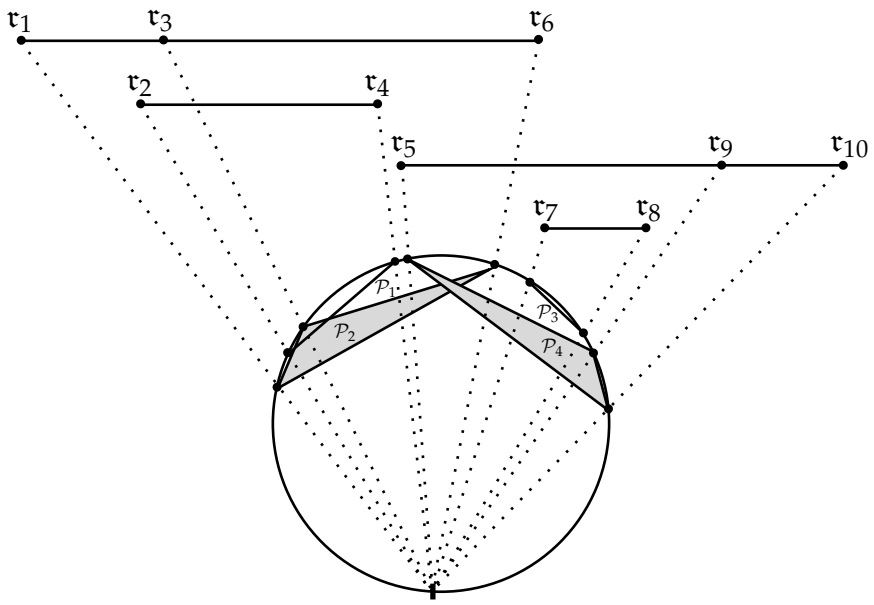


**Figure 2.3.** Projection of polygons to the real line

$\mathfrak{r}^i_{k_1} < \mathfrak{r}^j_{l_1} < \mathfrak{r}^i_{k_2} < \mathfrak{r}^j_{l_2}$ or with $\mathfrak{r}^j_{l_1} < \mathfrak{r}^i_{k_1} < \mathfrak{r}^j_{l_2} < \mathfrak{r}^i_{k_2}$. We will refer to this condition as *spider condition*. It is easy to see, that two vertices $\mathcal{P}_i(\mathfrak{R}_i)$ and $\mathcal{P}_j(\mathfrak{R}_j)$ in a polygon-circle graph are adjacent – that is, the polygons $\mathcal{P}_i(\mathfrak{R}_i)$ and $\mathcal{P}_j(\mathfrak{R}_j)$ intersect – if the two vertices $\mathfrak{R}_i$ and $\mathfrak{R}_j$ are adjacent in the respective spider graph – that is, the spider condition is true for $\mathfrak{R}_i$ and $\mathfrak{R}_j$ – and vice versa. Consequently, the class of polygon-circle graphs is equivalent to the class of spider graphs.

In the following we visualize the polygons by plotting the reals corresponding to the corner points of one polygon along a line between the reals of the first and the last corner point, see Figure 2.3. We refer to such a plot of a polygon as "*interval with intermediate points*", or IWIP for short. With a look at these IWIPs one can easily see by checking the spider condition whether or not two polygons intersect.

Note, we say polygon $\mathcal{P}_i(\mathfrak{R}_i)$ *spans* polygon $\mathcal{P}_j(\mathfrak{R}_j)$ or polygon $\mathcal{P}_j(\mathfrak{R}_j)$ *is spanned by* polygon $\mathcal{P}_i(\mathfrak{R}_i)$ if they do not intersect and if $\mathfrak{r}^i_1 < \mathfrak{r}^j_1 < \mathfrak{r}^j_{n^c_j} < \mathfrak{r}^i_{n^c_i}$. In the example shown in Figure 2.3, there is one spanned polygon: polygon $\mathcal{P}_4(\{\mathfrak{r}_5, \mathfrak{r}_9, \mathfrak{r}_{10}\})$ spans polygon $\mathcal{P}_3(\{\mathfrak{r}_7, \mathfrak{r}_8\})$.

Note that the polygons shown in Figure 2.4 – also see Figure 2.3 – intersect if and only if the corresponding curves that are plotted inter-

sect. This gives an intuition for the fact that each polygon-circle graph is an interval-filament graph.

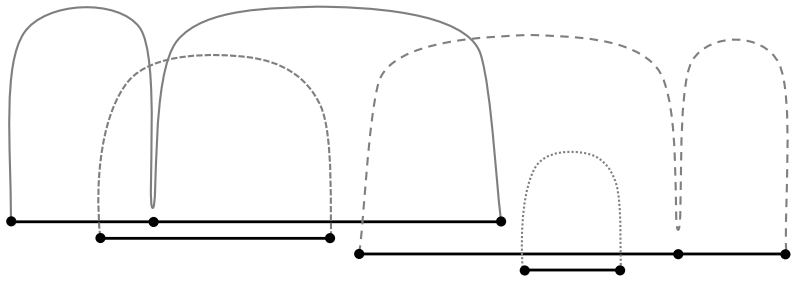

**Figure 2.4.** Relation between polygon-circle graphs and interval-filament graphs

GAVRIL (2000) introduced polynomial time algorithms for computing maximum weighted cliques – abbreviated MWC – and maximum weighted independent sets – MWIS for short – of interval-filament graphs. In the following, we briefly rephrase both algorithms for computing an MWC or an MWIS of a polygon-circle graph, because we apply them as subroutines for other problems, see Chapter 5. We reason the computational complexity of these algorithms; however, for the proofs of optimality we refer the reader to GAVRIL (2000).

### Maximum Weighted Cliques of Polygon-Circle Graphs

Given a polygon-circle graph $G^{PC}(\mathcal{P}) = (V, E)$ with $V = \{\mathcal{P}_1, \ldots, \mathcal{P}_{n^\mathcal{P}}\}$, let us define $V_i := \{\, \mathcal{P}_j \,|\, \mathcal{P}_j \in V, \mathfrak{r}_1^i \in \mathcal{I}^{\mathcal{P}_j}\}$.

**Algorithm 1:** MWC of a polygon-circle graph

**Input** : A polygon-circle graph $G^{PC}(\mathcal{P}) = (V, E)$ with $V = \{\mathcal{P}_1, \ldots, \mathcal{P}_{n^\mathcal{P}}\}$ and polygon weights $w_1, \ldots, w_{n^\mathcal{P}}$
**Output**: An MWC $C_{\text{wmax}}$ of $G^{PC}(\mathcal{P})$ with weight $w_{\text{wmax}}$

1 $\bar{w}_{\max} \leftarrow 0$
2 **for** $i \leftarrow 1$ **to** $n^\mathcal{P}$ **do**
3 Compute an MWIS $C^i_{\text{wmax}}$ of the graph $coG^{PC}[V_i]$ with weight $\bar{w}_i$
4 **if** $\bar{w}_i > \bar{w}_{\max}$ **then** $\bar{w}_{\max} \leftarrow \bar{w}_i$, $C_{\text{wmax}} \leftarrow C^i_{\text{wmax}}$

It is easily seen that the graphs $coG^{PC}[V_i]$ are comparability graphs with the following transitive orientation: edge $\{\mathcal{P}_i, \mathcal{P}_j\}$ is oriented from $i$ to $j$ if $\mathcal{I}^{\mathcal{P}_i} \subset \mathcal{I}^{\mathcal{P}_j}$. An MWIS of a transitive graph can be computed with the algorithm of KAGARIS & TRAGOUDAS (1999) in $\mathcal{O}(n^3)$. As a consequence, Algorithm 1 determines an MWC of a polygon-circle graph in $\mathcal{O}(n^4)$.

### Maximum Weighted Independent Sets of Polygon-Circle Graphs

Let a polygon representation $\tilde{\mathcal{P}} = \{\mathcal{P}_1, \ldots, \mathcal{P}_{n^{\mathcal{P}}}\}$ be given, see Convention 2.1. For ease of notation, we add a polygon $\mathcal{P}_0(\{0, M\})$ of weight $w_0 = 0$ to $\tilde{\mathcal{P}}$, where $M$ is a sufficiently large constant. Of course, we only need to remove $\mathcal{P}_0$ from an MWIS of $G^{PC}(\tilde{\mathcal{P}})$ to obtain an MWIS of $G^{PC}(\tilde{\mathcal{P}} \backslash \mathcal{P}_0)$. For Algorithm 2 – which determines an MWIS of such a polygon-circle graph $G^{PC}(\tilde{\mathcal{P}})$ – we denote the interval graph $G^I(\mathcal{I}^{\tilde{\mathcal{P}}})$ with $\mathcal{I}^{\tilde{\mathcal{P}}} = \{\, \mathcal{I}^{\mathcal{P}_i} \mid i = 0, 1, \ldots, n^{\mathcal{P}} \}$ by $\hat{G}^I$ for short and we define $\hat{V}_i = \{\, \mathcal{I}^{\mathcal{P}_j} \mid \text{polygon } \mathcal{P}_j \text{ is spanned by polygon } \mathcal{P}_i \}$.

**Algorithm 2:** MWIS of a polygon-circle graph

**Input** : A polygon-circle graph $G^{PC}(\tilde{\mathcal{P}}) = (V, E)$ with $V = \{\mathcal{P}_0(\{0, M\}), \mathcal{P}_1, \ldots, \mathcal{P}_{n^{\mathcal{P}}}\}$ and polygon weights $w_0 = 0, w_1, \ldots, w_{n^{\mathcal{P}}}$

**Output**: An MWIS $I_{\text{wmax}}$ of $G^{PC}(\tilde{\mathcal{P}})$ with weight $w_0$

1 **for** $i \leftarrow n^{\mathcal{P}}$ **to** 0 **do**
2 Compute an MWIS $I^i_{\text{wmax}}$ of the interval graph $\hat{G}^I[\hat{V}_i]$ with total weight $\hat{w}_i$ (the interval weights equal the polygon weights)
3 $w_i \leftarrow w_i + \hat{w}_i$
4 Insert pointers from polygon $\mathcal{P}_i$ to all polygons $\mathcal{P}_j \in I^i_{\text{wmax}}$
5 Construct $I_{\text{wmax}}$ by backtracking on the pointers from $I^0_{\text{wmax}}$

FRANK (1976) introduces an algorithm for determining an MWIS of a chordal graph which – given a perfect elimination order as input – runs in $\mathcal{O}(n)$ time. Since perfect elimination orders of chordal graphs are computable in $\mathcal{O}(n+m)$ time – see TARJAN & YANNAKAKIS (1984) – one can determine an MWIS of a chordal graphs, and particularly of an interval graph in $\mathcal{O}(n+m)$ time. As consequence, Algo-

rithm 2 produces an MWIS of a polygon-circle graph in $\mathcal{O}(n^2 + nm)$ time.

CHAPTER 3

# Mathematical Formulations and Relations

FOR **sequence** versions of SRS – in particular **sequential** or **concurrent** versions – an *input sequence* of $n$ units is described by an integer sequence $S = (s_1, \ldots, s_n)$ of $n$ elements. Here, the $i$-th unit ($i$-th element in $S$) belongs to the $s_i$-th group (integer $s_i$).

In **ordered** versions, the groups are numbered according to their departing order. For example, $S = (2,3,1,2,1,2,3)$ implies that the third and the fifth incoming unit form the first outgoing group; which is in most corresponding applications the first group within an outbound train assembled on one output track. In other words, the desired output sequence – denoted by $S^{\text{out}}$ – reads $(1,1,2,2,2,3,3)$.

Otherwise, in **free** versions, the groups may be arbitrarily numbered and these numbers do not carry any information on the ordering of the outgoing units.

As mentioned in Section 1.2, the $o$-**ordered** versions correspond to forming $o$ outbound trains on $o$ parallel output tracks. Let $g_{\hat{o}}$ denote the number of groups that leave within outbound train $\hat{o}$. An instance of an $o$-**ordered** version is given by an input sequence $S$ and numbers $g_1, \ldots, g_o$. Without loss of generality, we assume that the groups $(\sum_{j=1}^{\hat{o}-1} g_j) + 1, \ldots, (\sum_{j=1}^{\hat{o}-1} g_j) + g_{\hat{o}}$ leave with outbound train $\hat{o}$. That is, for an instance of a 2-**ordered** version with $g_1 = 3$ and $g_2 = 2$, the groups 1, 2, and 3 leave with outbound train 1, and the groups 4 and 5 leave with outbound train 2.

Obviously, the output sequence $S^{\text{out}}$ has the structure $o$-**ordered** $g$-**blocks** ($o$-**ordered** $g$-**pattern**) if it decomposes into subsequences $S_1^{\text{out}}, \ldots, S_o^{\text{out}}$, that is, $S^{\text{out}} = S_1^{\text{out}} \cup \cdots \cup S_o^{\text{out}}$, such that each subsequence

$S^{\text{out}}_{\hat{o}}$ has the structure **ordered $g$-blocks** (**ordered $g$-pattern**) with respect to the groups $(\sum_{j=1}^{\hat{o}-1} g_j)+1,\ldots,(\sum_{j=1}^{\hat{o}-1} g_j)+g_{\hat{o}}$. Note that the subsequence $S^{\text{out}}_{\hat{o}}$ corresponds to outbound train $\hat{o}$. For example, consider the instance given by the input sequence $S=(3,2,1,2,4,5,1,4)$, $g_1=3$, and $g_2=2$. Suppose that the units leave the sorting tracks in the order $(4,1,4,5,1,2,2,3)$. Obviously, this output sequence $S^{\text{out}}$ has the structure 2-**ordered** 5-**blocks** since it is possible to form the two outbound trains $(1,1,2,2,3)$ and $(4,4,5)$ on two parallel output tracks without additional rearrangements.

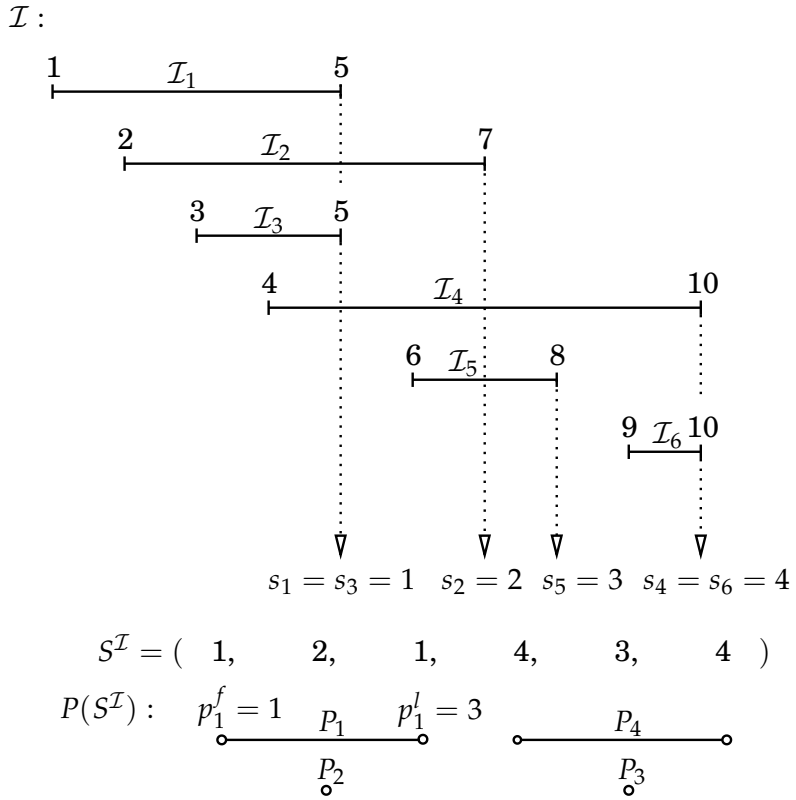


**Figure 3.1.** Time intervals and the corresponding input sequence

For **time windows**, **ordered** versions the input is more general than a sequence of integers. Instead, time intervals $\mathcal{I}_i=[a_i,d_i]\in\mathcal{I}$ between arrival time $a_i$ and departure time $d_i$ for each unit $i$ are given. We assume that no two units arrive at the same time and that the intervals are sorted by increasing arrival times, so the $i$-th unit arrives at position $i$ during the arrival process. A group is characterized by units having the same departure time, and the set of intervals that corre-

spond to group $\mathfrak{g}$ is denoted by $\mathcal{I}^{\mathfrak{g}}$. Primarily to enable comparisons – see Section 3.2 – we construct an integer sequence $S^{\mathcal{I}} = (s_1, \ldots, s_n)$ according to $\mathcal{I}$, that is, the $i$-th incoming unit leaves within the $s_i$-th departing group, see Figure 3.1. Note that within the sequence $S^{\mathcal{I}}$ some information of the exact arrival and departure times gets lost; for example, we could shift the intervals $\mathcal{I}_4$, $\mathcal{I}_5$, and $\mathcal{I}_6$ of the example in Figure 3.1 arbitrarily rightwards without noticing a change in $S^{\mathcal{I}}$.

For both **sequence** and **time windows** versions we consider two possible objectives in this thesis, cf. Section 1.2. Remember, in $t$-**minimizing** versions, the goal is to execute the sorting on a minimum number of tracks. This optimal value is denoted by $t^*_{\mathcal{V}_S}(S)$ for **sequence** versions $\mathcal{V}_S$ and by $t^*_{\mathcal{V}_\mathcal{I}}(\mathcal{I})$ for **time windows** versions $\mathcal{V}_\mathcal{I}$, abbreviated $t^*$. In $h$-**minimizing** versions which apply at rail yards featuring a hump, the goal is to perform the sorting with as few humping steps as possible occupying at most a given number of tracks. Here, the optimal value – that is, the minimum number of humping steps – is given by $h^*_{\mathcal{V}_S}(S)$, $h^*_{\mathcal{V}_\mathcal{I}}(\mathcal{I})$, or by $h^*$ for short.

## 3.1 Equivalent Formulations

### Sequence Partitioning

Most $t$-**minimizing** versions of SRS – in particular the **no shunting** versions – may be described as MINIMUM PARTITION (PROBLEMS) OF INTEGER SEQUENCES abbreviated MPIS, i. e., find a minimum *feasible* partition of $S$ or of $S^{\mathcal{I}}$ into subsequences $S_1, \ldots, S_{t^*}$ whose feasibility depends on the version, as well as on $\mathcal{I}$ in **time windows** versions. Note that each subsequence $S_t$ corresponds to the units placed on track $t$.

### Graph Coloring

For several $t$-**minimizing** versions $\mathcal{V}$ we can provide a condition that tells us whether or not any two units (groups) may be placed on the same track. For these versions $\mathcal{V}$, we are able to define a graph denoted by $G^{\mathcal{V}}$ whose vertices either are the units in **split** versions or are the groups in 0-**split** versions. Two vertices in these so-called $\mathcal{V}$-*graphs* are adjacent if and only if the two respective units (groups) may not be placed on the same track. In any feasible coloring of $G^{\mathcal{V}}$, the vertices

colored with color $i$ correspond to the units (groups) assigned to track $i$. For the rest of this section, we consider such versions $\mathcal{V}$ *with a* $\mathcal{V}$-graph.

**Minimum Vertex Coloring** If $\mathcal{V}$ is a $t$**-minimizing**, **unbounded** version it corresponds to MVC of the respective $\mathcal{V}$-graph, that is, $t^*_{\mathcal{V}}$ equals $\chi(G^{\mathcal{V}})$.

If $\mathcal{V}$ is a $t$**-minimizing**, $b$**-bounded** version it is equivalent to the problem of finding a coloring of the $\mathcal{V}$-graph with minimum number of colors, however, each color may not be used arbitrarily often.

**Mutual Exclusion Scheduling** If $\mathcal{V}$ is a $t$**-minimizing**, $b$**-bounded**, **split** version, at most $b$ vertices may be colored with the same color. As a consequence, such a version $\mathcal{V}$ is equivalent to $b$-MES regarding the respective class of $\mathcal{V}$-graphs. Note, 2-MES in an arbitrary graph $G$ is polynomially solvable by determining a MAXIMUM MATCHING in the complement of $G$. Hence, $t$**-minimizing**, 2**-bounded**, **split** versions $\mathcal{V}$ with a $\mathcal{V}$-graph are polynomially solvable.

**Bin Packing with Conflicts** If $\mathcal{V}$ is a $t$**-minimizing**, $b$**-bounded**, 0**-split** version, we may color the vertices $\mathfrak{g}_{i_1}, \ldots, \mathfrak{g}_{i_l}$ in the respective $\mathcal{V}$-graph with the same color, if $\sum_{j=1}^{l} \bar{n}_{\mathfrak{g}_{i_j}} \leq b$. Thus, such a version $\mathcal{V}$ corresponds to BPC regarding the conflict graph $G^{\mathcal{V}}$ for the items (groups) $1, \ldots, g$ with sizes $\bar{n}_1/b, \ldots, \bar{n}_g/b$. Note that for any of these versions $\mathcal{V}$, there always is an input sequence such that any two groups can be placed on the same track, that is, the conflict graph $G^{\mathcal{V}}$ is empty. Consequently, all $t$**-minimizing**, $b$**-bounded**, 0**-split** versions with a $\mathcal{V}$-graph generalize BIN PACKING, and cannot be approximated in polynomial time within a factor smaller than $3/2$, unless $\mathcal{P} = \mathcal{NP}$.

## 3.2 Relations between Versions

Some of the relations between versions of SRS presented in this section are rather trivial. Nevertheless, we list all of them as an overview.

As mentioned in the previous section, most $t$**-minimizing** versions of SRS are equivalent to finding a minimum feasible partition of the input sequence $S$. In the following we give an example.

Let us consider a $\underline{t}$**-st,**$\cdot$**|nsh,se,**$\cdot|\{$**fr,or**$\}$**,**$\cdot$ version, which is either a $\underline{t}$**-st,**$\cdot$**|nsh,se,**$\cdot|$**fr,**$\cdot$ version or a $\underline{t}$**-st,**$\cdot$**|nsh,se,**$\cdot|$**or,**$\cdot$ version. Here, a feasible schedule that requires $t$ tracks corresponds to a feasible partition of $S$ into subsequences $S_1, S_2, \ldots, S_t$ such that each subsequence $S_i$ again decomposes into $l_i$ subsequences, that is, $S_i = S_{i,1} \oplus S_{i,2} \oplus \cdots \oplus S_{i,l_i}$, see Figure 3.2. Each Subsequence $S_{i,j}$ corresponds to a block pulled out of the tracks without additional rearrangements at a time. Hence, we have $\bar{\mathfrak{q}} = \sum_{i=1}^{t} l_i$ pull outs in total; consequently, the output sequence is formed by consecutively connecting the $\bar{\mathfrak{q}}$ blocks (the element reversing subsequences), that is, $S^{\text{out}} = S_1^{\text{out}} \oplus S_2^{\text{out}} \oplus \cdots \oplus S_{\bar{\mathfrak{q}}}^{\text{out}}$. Moreover, a feasible schedule allows a chronology of pulling out the blocks which leads to the requested configuration of the output sequence. In mathematical terms, such an order corresponds to an assignment of the subsequences $S_{i,j}^{\leftrightarrow}$ to those of $S^{\text{out}}$ – that is, $S_{i,j}^{\leftrightarrow} \mapsto S_{\mathfrak{q}}^{\text{out}}$ – such that

$$j_2 > j_1 \ \wedge \ S_{i,j_1}^{\leftrightarrow} \mapsto S_{\mathfrak{q}_1}^{\text{out}} \ \wedge \ S_{i,j_2}^{\leftrightarrow} \mapsto S_{\mathfrak{q}_2}^{\text{out}} \quad \Rightarrow \quad \mathfrak{q}_2 < \mathfrak{q}_1$$

and that $S^{\text{out}}$ has the desired structure. In respective 0**-split** versions, we additionally require that all units of the same group belong to the same subsequence $S_i$. In $b$**-bounded** versions the cardinality of each $S_i$ is bounded by $b$. On the contrary, if a schedule does not correspond to a partition possessing the mentioned qualities, it is not feasible. Thus, any $\underline{t}$**-st,**$\cdot$**|nsh,se,**$\cdot|\{$**fr,or**$\}$**,**$\cdot$ version is equivalent to finding a minimum partition of $S$ with the above properties.

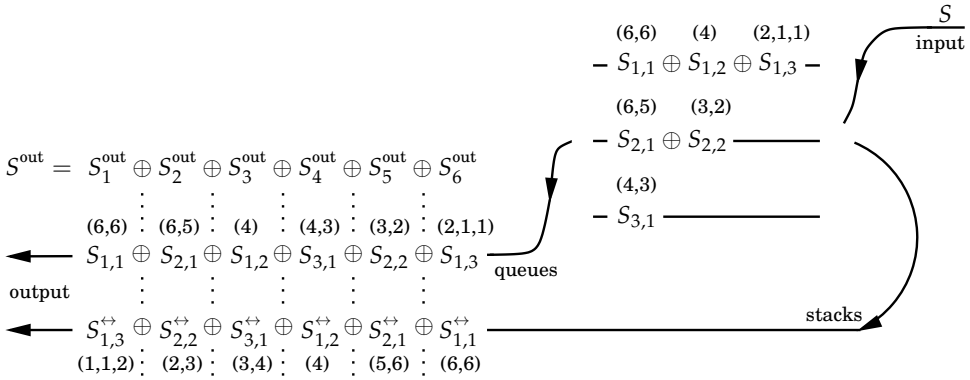


**Figure 3.2.** Relation between the $\underline{t}$**-st,**$\cdot$**|nsh,se,**$\cdot|\{$**fr,or**$\}$**,**$g$**-bl** versions and the corresponding **queues** versions

A closer look at Figure 3.2 reveals the connection between the $\underline{t}$**-st,**$\cdot$**|nsh,se,**$\cdot|\{$**fr,or**$\}$**,**$g$**-bl** versions and the corresponding **queues** versions.

Among such pairs, consider a couple of **free** versions. Suppose we had the possibility to use the tracks either as **stacks** or as **queues** and we know a solution for the **stacks** case. By reversing the chronological order of pulling out the blocks given by this schedule it is possible to form an output sequence by consecutively connecting the subsequences $S_{i,j}$ using the tracks as **queues**. In the resulting output sequence the units occur in reversed order as in the **stacks** case. Clearly, if $S$ has the structure **free** $g$**-blocks** then $S^{\leftrightarrow}$ also does. Hence, each solution of a $\underline{t}$**-st,**$\cdot|$**nsh,se,**$\cdot|$**fr,**$g$**-bl** version requiring $t$ tracks is a solution of the respective $\underline{t}$**-qu,**$\cdot|$**nsh,se,**$\cdot|$**fr,**$g$**-bl** version occupying the same number of tracks, and vice versa.

**Observation 3.1**
*Each* $\underline{t}$***-st,***$\cdot|$***nsh,se,***$\cdot|$***fr,***$g$***-bl** version is equivalent to the respective* $\underline{t}$***-qu,***$\cdot|$***nsh,se,***$\cdot|$***fr,***$g$***-bl** version.*

Let us now switch to the respective **ordered** versions. In general, a reversed pull out of the blocks described by a solution for the **stacks** case does not lead to the desired output sequence for the respective **queues** case – we actually need it back to front – see Figure 3.2. However, the following observation is easily seen.

**Observation 3.2**
*We can compute an optimal solution of a* $\underline{t}$***-qu,***$\cdot|$***nsh,se,***$\cdot|$***or,***$g$***-bl** version with input sequence $S$ by solving the respective* $\underline{t}$***-st,***$\cdot|$***nsh,se,***$\cdot|$***or,***$g$***-bl** version with the input sequence* $S^{\updownarrow}$*, and vice versa.*

This property extends to the respective $o$**-ordered** versions for forming $o$ outbound trains on parallel output tracks.

**Observation 3.3**
*An optimal solution of a* $\underline{t}$***-qu,***$\cdot|$***nsh,se,***$\cdot|$***o-or,***$g$***-bl** version with input sequence $S$ is computable by solving the corresponding **stacks** version* $\underline{t}$***-st,***$\cdot|$***nsh,se,***$\cdot|$***o-or,***$g$***-bl** with the input sequence* $S^{\updownarrow}$*, and vice versa.*

We leave the question whether the computational complexity of any $\underline{t}$**-st,**$\cdot|$**nsh,**$\cdot$**,**$\cdot|\cdot$**,**$g$**-bl** version equals the computational complexity of the corresponding **queues** version $\underline{t}$**-qu,**$\cdot|$**nsh,**$\cdot$**,**$\cdot|\cdot$**,**$g$**-bl** open for the moment. It is answered later in Subsection 4.3.1.

Now, let us consider **unbounded**, $o$**-ordered** versions. In all $o$**-ordered** versions the input sequence $S$ decomposes into $o$ subsequences $S^1, \ldots, S^o$ – that is, $S = S^1 \cup \cdots \cup S^o$ – such that $S^{\hat{0}}$ contains

all units leaving with outbound train $\hat{o}$, that is, all units that belong to the groups $(\sum_{j=1}^{\hat{o}-1} g_j)+1,\ldots,(\sum_{j=1}^{\hat{o}-1} g_j)+g_{\hat{o}}$. We define $n_{\hat{o}}$ as the length of subsequence $S^{\hat{o}}$. For example, for the instance of a 2-**ordered** version given by the input sequence $S=(3,2,1,2,4,5,1,4)$, $g_1=3$, and $g_2=2$, we get $S^1=(3,2,1,2,1)$ of length $n_1=5$ and $S^2=(4,5,4)$ of length $n_2=3$.

**Observation 3.4**
*We can determine an optimal schedule for an* ***unbounded****,* $o$*-****ordered*** *version by "*gluing*" together the* $o$ *optimal schedules for the corresponding* ***ordered*** *version obtained for the input sequences* $S^1,\ldots,S^o$*.*

For $t$-**minimizing**, **no shunting** versions let us analyze the effect of the distinguished cases for the chronological order of arrival and departure – **sequential**, **concurrent**, and **time windows** – on the minimum number $t^*$ of tracks required for the sorting of an instance given by $\mathcal{I}$ and corresponding $S^{\mathcal{I}}$. By definition, the times the units may leave the tracks at departure are not increasing when switching from a **sequential** version to the respective **time windows** version, and again when switching from a **time windows** version to the respective **concurrent** version.

For **stacks** versions, we gain some freedom and potential for improvements when switching from a **sequential** version to the respective **time windows** version, and also when switching from a **time windows** version to the respective **concurrent** version. Hence, we observe the following inequalities.

**Observation 3.5**
*It holds for the respective versions*

$$t^*_{\underline{t}\text{-}st,\cdot|nsh,se,\cdot|\cdot,\cdot}(S^{\mathcal{I}}) \quad\geq\quad t^*_{\underline{t}\text{-}st,\cdot|nsh,tw,\cdot|\cdot,\cdot}(\mathcal{I}) \quad\geq\quad t^*_{\underline{t}\text{-}st,\cdot|nsh,co,\cdot|\cdot,\cdot}(S^{\mathcal{I}}) \quad .$$

Now, let us consider **queues** versions. Here, in any of the cases **sequential**, **concurrent**, and **time windows**, a unit $j$ cannot be placed on the same track behind a unit $i$ if $j$ has to depart earlier than $i$, due to the order-presuming behavior of queues.

**Observation 3.6**
*A* $\underline{t}$***-qu,***$\cdot|$***nsh,se,***$\cdot|\cdot,\cdot$ *version and the respective* $\underline{t}$***-qu,***$\cdot|$***nsh,tw,***$\cdot|\cdot,\cdot$ *and* $\underline{t}$***-qu,***$\cdot|$***nsh,co,***$\cdot|\cdot,\cdot$ *versions are pairwise equivalent. Thus,*

$$t^*_{\underline{t}\text{-}qu,\cdot|nsh,se,\cdot|\cdot,\cdot}(S^{\mathcal{I}}) \quad=\quad t^*_{\underline{t}\text{-}qu,\cdot|nsh,tw,\cdot|\cdot,\cdot}(\mathcal{I}) \quad=\quad t^*_{\underline{t}\text{-}qu,\cdot|nsh,co,\cdot|\cdot,\cdot}(S^{\mathcal{I}}) \quad .$$

The following inequalities trivially hold since solutions of the 0-**split** versions remain feasible for the respective **split** and **chain-split** versions.

**Observation 3.7**
*It holds for the respective versions*

$$t^*_{\cdot,\cdot|\cdot,\cdot,0\text{-}sp|\cdot,\cdot} \geq t^*_{\cdot,\cdot|\cdot,\cdot,sp|\cdot,\cdot} \quad ,$$

$$t^*_{\cdot,\cdot|nsh,se,0\text{-}sp|\cdot,g\text{-}bl} \geq t^*_{\cdot,\cdot|nsh,se,csp|\cdot,g\text{-}bl} \quad .$$

Since solutions of the **ordered** versions are also feasible for the respective **free** versions, we get the following inequalities.

**Observation 3.8**
*It holds for the respective versions*

$$t^*_{\cdot,\cdot|\cdot,\cdot,\cdot|or,\cdot} \geq t^*_{\cdot,\cdot|\cdot,\cdot,\cdot|fr,\cdot} \quad .$$

Finally, each solution for a $b$-**bounded** version remains feasible assuming unbounded track length.

**Observation 3.9**
*It holds for the respective versions*

$$t^*_{\cdot,b\text{-}bd|\cdot,\cdot,\cdot|\cdot,\cdot} \geq t^*_{\cdot,ub|\cdot,\cdot,\cdot|\cdot,\cdot} \quad .$$

For some $t$-**minimizing** $\cdot$**,ub**|**nsh,**$\cdot$,$\cdot$|$\cdot$**,**$g$-**bl** versions, Figure 3.3 illustrates optimal schedules for the instance given in Figure 3.1.

In view of computational complexity, we list versions that generalize other versions.

Of course, for general $b$, a $b$-**bounded** version is a generalization of the corresponding **unbounded** version.

**Observation 3.10**
*The $\mathcal{NP}$-hardness of an* ***unbounded*** *version extends to the corresponding $b$-**bounded** version for general $b$.*

An $n$-**blocks** version is a special case of the respective $g$-**blocks** version. Note, $n$-**blocks** equals $n$-**pattern**.

**Observation 3.11**
*The $\mathcal{NP}$-hardness of an $n$-**blocks** version extends to the corresponding $g$-**blocks** version.*

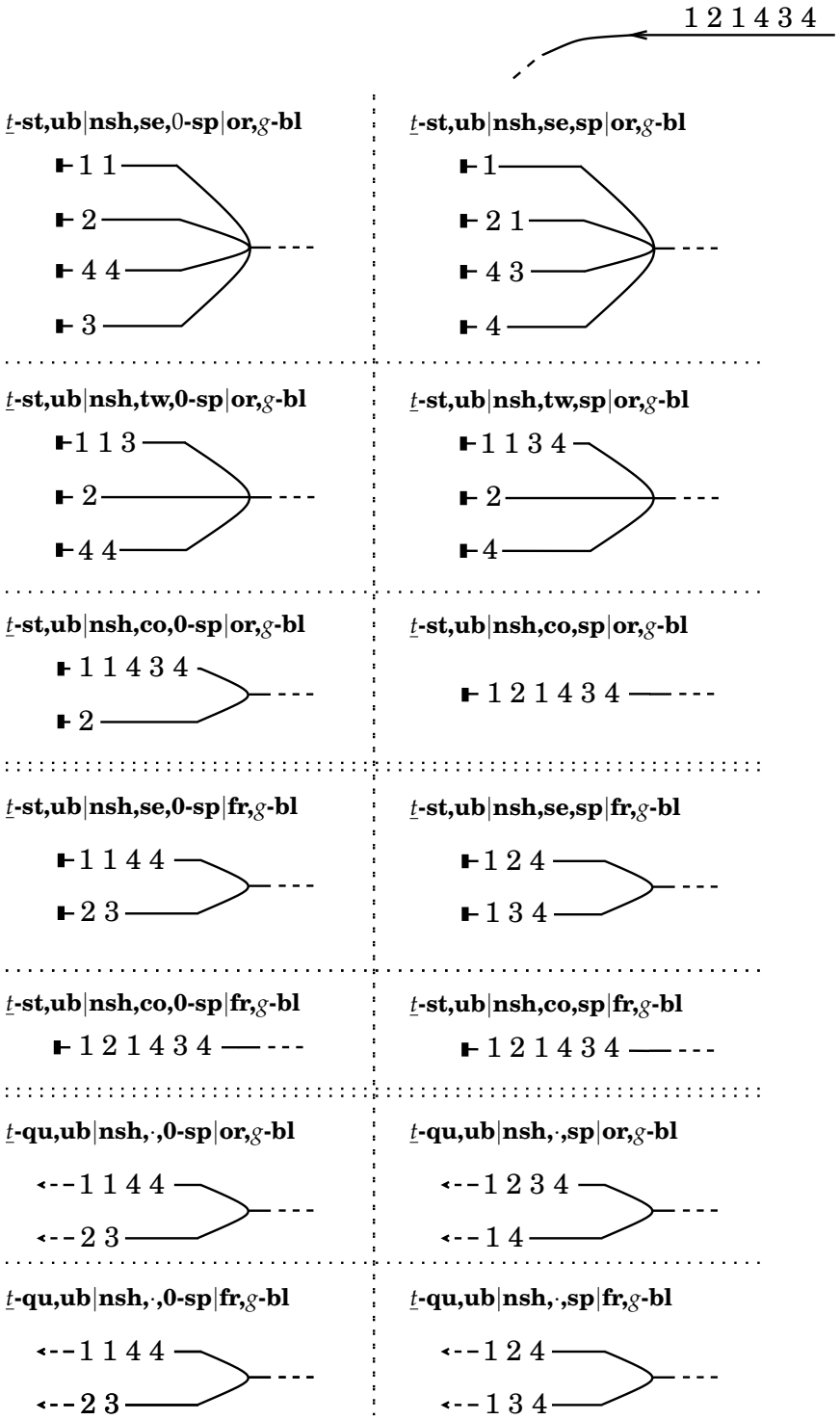


**Figure 3.3.** Optimal schedules for the instance given in Figure 3.1 for some *t*-**minimizing** ·,**ub**|**nsh**,·,·|·,*g*-**bl** versions

An $o$**-ordered** version generalizes the corresponding **ordered** version.

**Observation 3.12**
*The $\mathcal{NP}$-hardness of an **ordered** version extends to the corresponding $o$-**ordered** version.*

For a sequence, the structure **ordered** $g$**-blocks** is a particular **ordered** $g$**-pattern**. Thus, an **ordered** $g$**-pattern** version is generalization of the respective **ordered** $g$**-blocks** version.

**Observation 3.13**
*The $\mathcal{NP}$-hardness of an **ordered** $g$**-blocks** version extends to the corresponding **ordered** $g$**-pattern** version.*

In a **free** $g$**-blocks** version we actually allow several particular **free** $g$**-patterns** for the output sequence. For example, for the input sequence $S = (1,2,1,3)$ one of the three **free** 3**-patterns** $(a,a,b,c)$, $(a,b,b,c)$, and $(a,b,c,c)$ is a feasible structure of the output sequence. However, if we restrict the **free** $g$**-blocks** version to instances $\bar{\mathfrak{I}}$ where each group has the same cardinality, that is, $\bar{n}_{\mathfrak{g}_i} = \bar{n}_{\mathfrak{g}_j}$ for $1 \leq i < j \leq g$, then there is only one particular feasible **free** 3**-pattern** for the output sequence. For example, this pattern reads $(aa,bb,cc)$ given the input sequence $(2,1,3,2,3,1)$. As a consequence, **free** $g$**-pattern** (as well as **free** $g$**-blocks**) versions generalize the corresponding **free** $g$**-blocks** versions restricted to instances $\bar{\mathfrak{I}}$.

**Observation 3.14**
*The $\mathcal{NP}$-hardness of a **free** $g$**-blocks** version that are restricted to instances $\bar{\mathfrak{I}}$ defined above extends to the corresponding **free** $g$**-pattern** version for arbitrary instances.*

## 3.3 Related Problems

The problem of rearranging railcars was a founding inspiration in theoretical studies of permutations and integer sequences. Motivated by this application, TARJAN (1972), KNUTH (1973), and PRATT (1973) launched investigations that opened the door to many interesting theoretical questions.

For example, in the field of computer science, the theory of *sortable permutations* is concerned with the following questions. Which structure of permutations is required such that they are sortable with a

particular sorting device? In other words, is it possible to characterize the class of sortable permutations by a finite set of forbidden subsequences? Besides that, given a certain sorting device, how many sortable permutations of a given length do exist? In his book, BONA (2004) summarizes corresponding answers and results; for recent work see for example SMITH & VATTER (2009), ALDRED ET AL. (2010), and ALBERT ET AL. (2010).

Besides a certain connection of SRS to the above counting problems, SRS is strongly related to the optimization problems MWIS, MVC, MES, and BPC, see Section 3.1. Of course, there is a broad range of literature dealing with these well-known problems. At this point, we do not go into details; we will refer to relevant publications and results throughout the following chapters.

It follows a list of other real-world applications that have been tackled by mathematical optimization approaches and that – from a theoretical point of view – are similar to SRS.

- dispatching buses in parking depots,
  see GALLO & DI MIELE (2001) and HAMDOUNI ET AL. (2007),
- stowage planning of containers in container ships,
  see AVRIEL ET AL. (2000),
- storage planning of steel slabs in integrated steel production,
  see KÖNIG ET AL. (2007), KÖNIG & LÜBBECKE (2008), and KÖNIG (2009),
- job sequencing on conveyor or automated storage systems,
  see HAN (2004) and DEMANGE ET AL. (2009).

CHAPTER 4

# Computational Complexity

In this chapter we classify the computational complexity of many SRS versions. For the versions listed in Table 1.1, we summarize known results in the literature. In addition, we provide new complexity results; in particular, for versions relating to the real-world optimization problem considered in our project together with BASF. We present algorithms for relevant polynomially solvable versions, and for a few $\mathcal{NP}$-hard versions approximability results are given.

## 4.1 Permutation and Pattern Versions

Note that for the equivalent cases $n$**-blocks** and $n$**-pattern** the input sequence is a permutation and the splitting conditions coincide.

### 4.1.1 Unbounded Case

Di Stefano & Koči (2004) present various results for $n$**-blocks** versions. They show that the equivalent versions $\underline{t}$**-st,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** and $\underline{t}$**-qu,ub**|**nsh,**$\cdot$**,**$\cdot$|**or,**$n$**-bl** correspond to MVC of permutation graphs. As a consequence, these versions are solvable in $\mathcal{O}(n \log n)$ time. It is also mentioned that the version $\underline{t}$**-st,ub**|**nsh,tw,**$\cdot$|**or,**$n$**-bl** is equivalent to MVC of circle graphs; consequently, it is $\mathcal{NP}$-hard. Furthermore, the authors introduce minimum vertex coloring formulations in different hypergraphs for the versions $\underline{t}$**-dd,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl**, $\underline{t}$**-sd,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl**, and $\underline{t}$**-ds,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl**. The latter two versions are shown to be equivalent. Another result is that $t$**-sd,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** corresponds to deciding whether there exists

a partition of the given permutation into at most $t$ upper unimodal subsequences, which is proven to be $\mathcal{NP}$-complete by DI STEFANO ET AL. (2006). For the $\mathcal{NP}$-hard versions $\underline{t}$**-sd,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** and $\underline{t}$**-ds,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl**, DI STEFANO & KOČI (2004) suggest a greedy strategy that computes solutions with at most $\lfloor(\sqrt{8n+1}-1)/2\rfloor$ tracks in $\mathcal{O}(n^{2.5})$ time. This bound can be reasoned by the following fact which is mentioned by CHUNG (1980). Any permutation of length $n$ contains an upper unimodal sequence of length at least $\lceil\sqrt{2n+1/4}-1/2\rceil$, see STEELE (1995) for a probabilistic analysis. Recently, the versions $\underline{t}$**-sd,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** and $\underline{t}$**-ds,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** were shown to be 2-approximable in polynomial time by applying an LP rounding technique, cf. DI STEFANO ET AL. (2008). Finally, DI STEFANO & KOČI (2004) illustrate that the version $\underline{t}$**-dd,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** is equivalent to finding a minimum partition of the given permutation into unimodec subsequences, which is proven to be $\mathcal{NP}$-hard and 3-approximable in polynomial time by DI STEFANO ET AL. (2008).

It is well-known that the version $\underline{t}$**-sq,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** is equivalent to finding a minimum partition of the given permutation into monotone subsequences. For this version, an upper bound on the minimum number of occupied tracks can be derived from a classical result of ERDŐS & SZEKERES (1935). It says that any permutation of length $n$ contains a monotone subsequence of length $\lceil\sqrt{n}\,\rceil$. By recursive extraction of longest monotone sequences one can partition a permutation of length $n$ into at most $2\lfloor\sqrt{n}\rfloor$ monotone subsequences in $\mathcal{O}(n^{1.5})$ time, cf. BAR-YEHUDA & FOGEL (1998). In other words, a feasible schedule for above version with at most $2\lfloor\sqrt{n}\rfloor$ tracks is quickly obtainable. After all, the version $\underline{t}$**-sq,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** – which is equivalent to MIN COCOLORING of permutation graphs – is proven to be $\mathcal{NP}$-hard in WAGNER (1984), and it is 1.71-approximable in polynomial time, see FOMIN ET AL. (2002). However, one can determine a feasible schedule which exactly occupies $l$ stacks and $m$ queues in the rail yard in $\mathcal{O}(n^{l+m})$ time in case such a schedule exists, see BRANDSTÄDT & KRATSCH (1986).

### 4.1.2 Bounded Case

In WINTER (2000) the $\mathcal{NP}$-hardness of $\underline{t}$**-st,**$b$**-bd**|**nsh,se,sp**|**or,**$g$**-pa** for fixed $b \geq 4$ is shown. A more comprehensive version of the proof is published in WINTER & ZIMMERMANN (2000). Recently, EGGERMONT ET AL. (2009) proved that the version $\underline{t}$**-st,**$b$**-bd**|**nsh,se,sp**|**or,**$g$**-pa** is

$\mathcal{NP}$-hard, even for $b = 3$. Moreover, WINTER (2000) introduces a polynomial time algorithm for solving a version $\underline{t}$**-st,**2**-bd**|**nsh,se,**$\cdot$|**or,**$n$**-bl**. We improve this result by introducing a linear time algorithm for the generalization $\underline{t}$**-st,**$b$**-bd**|**nsh,se,**$\cdot$|**or,**$n$**-bl**, see Corollary 4.2.

## 4.2 Chain-Split Versions

Remember, in **chain-split** versions the units have to be placed on the tracks in such a way that pulling out the units track by track leads to the output sequence with the desired configuration. Note that **chain-split** is reasonable only for the **sequential** and **no shunting** case.

### 4.2.1 Unbounded Case

The version $\underline{t}$**-qu,ub**|**nsh,se,csp**|**or,**$g$**-bl** ($\underline{t}$**-qu,ub**|**nsh,se,csp**|**fr,**$g$**-bl**) corresponds to finding a minimum partition of the input sequence $S$ into subsequences $S_1, \ldots, S_{z^*}$ such that the output sequence $S_1 \oplus S_2 \oplus \cdots \oplus S_{z^*}$ has the structure **ordered** $g$**-blocks** (**free** $g$**-blocks**). Note, to ensure the structure **ordered** $g$**-blocks** at departure the tracks have to be emptied in order of increasing track numbers which correspond to the indices of the subsequences.

It is quite evident, that the version $\underline{t}$**-qu,ub**|**nsh,se,csp**|**or,**$g$**-bl** can be solved by iteratively constructing the subsequences with a greedy strategy. For example, let be given $S = (2,3,4,3,1,2,4)$. Then the best choice for $S_1$ obviously is $(1,2)$, a brief look at the remaining sequence $S \backslash S_1 = (2,3,4,3,4)$ reveals that $(2,3,3,4)$ is best possible for $S_2$, and finally we get with $S_3 = (4)$ a minimum partition into three subsequences. DAHLHAUS ET AL. (2000b) provide an algorithm which implements this greedy strategy in linear time, see Algorithm 3 for a slightly modified version.

This implementation maintains for each integer $\mathfrak{g} \in \mathcal{G}^S$ pointers First[ $\mathfrak{g}$ ] and Last[ $\mathfrak{g}$ ] to the first and last element of integer $\mathfrak{g}$ in $S_{n,g}$ as well as for each element $i$ pointers Next[ $i$ ], NextOfNext[ $i$ ], and PrevOfNext[ $i$ ] to the "*next*" element of the same integer $s_i$, to the next element of integer $s_i + 1$, and to the "*previous*" element of integer $s_i + 1$, respectively. For above example, these pointers read First=$(5,1,2,3)$, Last=$(5,6,4,7)$, Next=$(6,4,7,0,0,0,0)$, NextOfNext=$(2,3,0,7,6,0,0)$, and PrevOfNext=$(0,0,0,3,1,4,0)$. Figure 4.1 illustrates the none-zero

Next-, NextOfNext-, and PrevOfNext-pointers after initialization (lines 1–12 in Algorithm 3).

**Algorithm 3:** Greedy for $\underline{t}$**-qu,ub|nsh,se,csp|or,**$g$**-bl**

**Input** : An integer sequence $S_{n,g}$
**Output**: A minimum feasible partition of $S_{n,g}$ into subsequences in $\mathcal{O}(n)$ time

```
1  Next [ i ] ← NextOfNext [ i ] ← PrevOfNext [ i ] ← 0,    i = 1, ..., n
2  First [ i ] ← Last [ i ] ← Tmp [ i ] ← 0,    i = 1, ..., g
3  for i ← 1 to n do
4      if Last [ s_i ] = 0 then First [ s_i ] ← i
5      else Next [ Last [ s_i ] ] ← i
6      if s_i < g and Last [ s_i + 1 ] ≠ 0 then
7          PrevOfNext [ i ] ← Last [ s_i + 1 ]
8      Last [ s_i ] ← i
9  for i ← 1 to n − 1 do
10     if s_i < g then
11         if Tmp [ s_i + 1 ] = 0 then NextOfNext [ i ] ← First [ s_i + 1 ]
           else NextOfNext [ i ] ← Next [ Tmp [ s_i + 1 ] ]
12     Tmp [ s_i ] ← i
13 c ← s_0 ← First [ 0 ] ← SubSeq [ 0 ] ← 0
14 firstOfNextSub ← First [ 1 ]
15 while firstOfNextSub ≠ 0 do
16     c ← c + 1,  i ← firstOfNextSub,  firstOfNextSub ← 0
17     while i ≠ 0 do
18         SubSeq [ i ] ← c,  tmpi ← i,  i ← Next [ i ]
19         if SubSeq [ i ] > 0 or ( i = 0 and firstOfNextSub = 0 )
           then
20             i ← NextOfNext [ tmpi ]
21             firstOfNextSub ← First [ s_PrevOfNext[ tmpi ] ]
```

It is easily proven that Algorithm 3 has a time complexity of $\mathcal{O}(n)$, that is, version $\underline{t}$**-qu,ub|nsh,se,csp|or,**$g$**-bl** is solvable in linear time. Hence, $\underline{t}$**-st,ub|nsh,se,csp|or,**$g$**-bl** is also solvable in linear time by applying Algorithm 3 to the integer reversing sequence $S^{\updownarrow}$ of the input sequence $S$, see Observation 3.2.

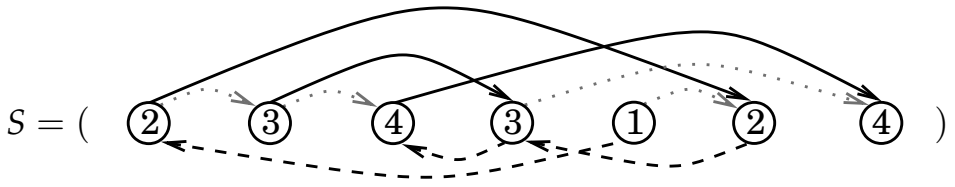


**Figure 4.1.** None-zero pointers Next (solid), NextOfNext (dotted), and PrevOfNext (dashed) for the sequence $S = (2,3,4,3,1,2,4)$ after initialization in Algorithm 3

The version $\underline{t}$**-qu,ub**|**nsh,se,csp**|**fr,**$g$**-bl** is shown to be $\mathcal{NP}$-hard by DAHLHAUS ET AL. (2000a). Consequently, $\underline{t}$**-st,ub**|**nsh,se,csp**|**fr,**$g$**-bl** is also $\mathcal{NP}$-hard, see Observation 3.1.

**Theorem 4.1**
*The versions* $\underline{t}$***-qu,ub*|*nsh,se,csp*|*fr,*$g$***-bl**, $\underline{t}$***-st,ub*|*nsh,se,csp*|*fr,*$g$***-bl** *are 2-approximable in* $\mathcal{O}(n \log n)$ *time.*

*Proof.* For version $\underline{t}$**-qu,ub**|**nsh,se,csp**|**fr,**$g$**-bl** consider the optimal solution occupying $t^*_{\text{csp}}$ tracks. Clearly, at most $t^*_{\text{csp}} - 1$ groups are split up over two tracks. By moving any group that is split up to an additional track we get a feasible solution for version $\underline{t}$**-qu,ub**|**nsh,se,**0**-sp**|**fr,**$g$**-bl** with at most $2 \cdot t^*_{\text{csp}} - 1$ tracks. The optimal solution of $\underline{t}$**-qu,ub**|**nsh,se,**0**-sp**|**fr,**$g$**-bl** with $t^*_{\text{nsp}} \leq 2 \cdot t^*_{\text{csp}} - 1$ tracks is computable in $\mathcal{O}(n \log n)$ time, see Subsection 4.3.1. □

Note that a complete enumeration of the optimal schedules of version $\underline{t}$**-st,ub**|**nsh,se,csp**|**or,**$g$**-bl** for all possible departing orders of the groups takes $\mathcal{O}(g! \cdot n)$ time. By selecting the best of these solutions we obtain an optimal solution of version $\underline{t}$**-st,ub**|**nsh,se,csp**|**fr,**$g$**-bl**. That may be a reasonable strategy for a very small number $g$ of groups. Analogously, we obtain an optimal schedule for the version $\underline{t}$**-qu,ub**|**nsh,se,csp**|**fr,**$g$**-bl** in $\mathcal{O}(g! \cdot n)$ time. Hence, for fixed $g$, the versions $\underline{t}$**-st,ub**|**nsh,se,csp**|**fr,**$g$**-bl** and $\underline{t}$**-qu,ub**|**nsh,se,csp**|**fr,**$g$**-bl** can be solved in polynomial time.

### 4.2.2 Bounded Case

The $\underline{t}$**-qu,**$b$**-bd**|**nsh,se,csp**|**fr,**$g$**-bl** and $\underline{t}$**-st,**$b$**-bd**|**nsh,se,csp**|**fr,**$g$**-bl** versions are $\mathcal{NP}$-hard since already the respective **unbounded** versions are $\mathcal{NP}$-hard. Together with a diploma student we obtained a linear time algorithm for optimally solving the respective 2**-bounded** versions $\underline{t}$**-qu,**2**-bd**|**nsh,se,csp**|**fr,**$g$**-bl** and $\underline{t}$**-st,**2**-bd**|**nsh,se,csp**|**fr,**$g$**-bl**, see SPIERLING (2007).

**Algorithm 4:** Greedy for $\underline{t}$**-qu,**$b$**-bd**|**nsh,se,csp**|**or,**$g$**-bl**

**Input** : An integer sequence $S_{n,g}$
**Output**: A minimum feasible partition of $S_{n,g}$ into subsequences in $\mathcal{O}(n)$ time

```
1  while firstOfNextSub ≠ 0 do
2      c ← c + 1,  i ← firstOfNextSub,  firstOfNextSub ← 0
3      subSize ← 0
4      while i ≠ 0 do
5          SubSeq [ i ] ← c,  subSize ← subSize + 1
6          if subSize = b then
7              if firstOfNextSub ≠ 0 then
8                  Next [ tmpPrevOfNext ] ← Next [ i ]
9              else
10                 if Next [ i ] ≠ 0 and SubSeq [ Next [ i ] ] = 0 then
                   firstOfNextSub ← Next [ i ]
11                 else if First [ s_PrevOfNext[ i ] ] ≠ 0 then
                   firstOfNextSub ← First [ s_PrevOfNext[ i ] ]
12                 else if SubSeq [ NextOfNext [ i ] ] = 0 then
                   firstOfNextSub ← NextOfNext [ i ]
13             i ← 0
14         else
15             tmpi ← i,  i ← Next [ i ]
16             if SubSeq [ i ] > 0 or ( i = 0 and firstOfNextSub = 0 )
               then
17                 i ← NextOfNext [ tmpi ]
18                 firstOfNextSub ← First [ s_PrevOfNext[ tmpi ] ]
19                 tmpPrevOfNext ← PrevOfNext [ tmpi ]
```

In view of version $\underline{t}$**-qu,**$b$**-bd|nsh,se,csp|or,**$g$**-bl**, let us consider the example of the previous subsection, that is, the input sequence $S = (2,3,4,3,1,2,4)$. The optimal solution $S_1 = (1,2)$, $S_2 = (2,3,3,4)$, and $S_3 = (4)$ for the **unbounded queues** case – determined with the greedy strategy described by Algorithm 3 – loses its feasibility if only three units are allowed to be placed on the same track. However, an adaption of the greedy such that it stops assigning units to the current track whenever the capacity of this track is reached computes an optimal solution for the **bounded** case, see Algorithm 4. The optimal solution for our example reads $S_1 = (1,2)$, $S_2 = (2,3,3)$, and $S_3 = (4,4)$ in case of **3-bounded queues**. Note, the initializing of the pointers is the same as in Algorithm 3 (lines 1–14). The optimality and the linearity of Algorithm 4 are easily seen.

**Corollary 4.1**
*Version* $\underline{t}$***-qu,***$b$***-bd|nsh,se,csp|or,***$g$***-bl** is solvable in $\mathcal{O}(n)$ time by applying Algorithm 4 to the input sequence $S$.*

For the respective **b-bounded stacks** version we get the following conclusion, see Observation 3.2.

**Corollary 4.2**
*Version* $\underline{t}$***-st,***$b$***-bd|nsh,se,csp|or,***$g$***-bl** is solvable in $\mathcal{O}(n)$ time by applying Algorithm 4 to the integer reversing sequence $S^{\updownarrow}$ of the input sequence $S$.*

## 4.3 Versions with no Shunting

This section deals with **no shunting**, $g$**-blocks** versions.

Remember, in 0**-split** versions we need to park all units of one group on the same track. We say *we park a group* $\mathfrak{g}$ *on track* $t$ instead of we place all units of group $\mathfrak{g}$ on track $t$. For various SRS versions, the following theorem provides conditions telling us whether two groups in 0**-split** versions or two units in **split** versions may be parked on the same track without causing additional rearrangements at departure.

**Theorem 4.2**
*In case of* ***unbounded*** *tracks two units $i$ and $j$ in* ***split*** *versions or two groups $u$ and $v$ in* 0***-split*** *versions with $u < v$ cannot be parked on the same track if and only if the input sequence $S$ in* ***sequence*** *versions*

*or $S^{\mathcal{I}}$ in* ***time windows*** *versions contains a so-called* forbidden *subsequence. For various* ***unbounded***, ***no shunting***, $g$***-blocks*** *versions the structure of forbidden subsequences is listed in the table below.*

| *Version* | *Forbidden subsequence of $S$ (or of $S^{\mathcal{I}}$ )* | |
|---|---|---|
| $t$**-st,ub\|nsh,se,sp\|or,**$g$**-bl** | $(s_i, s_j)$ | *with* $s_i < s_j$ |
| $t$**-qu,ub\|nsh,se,sp\|or,**$g$**-bl** | $(s_i, s_j)$ | *with* $s_i > s_j$ |
| $t$**-qu,ub\|nsh,co,sp\|or,**$g$**-bl** | $(s_i, s_j)$ | *with* $s_i > s_j$ |
| $t$**-qu,ub\|nsh,tw,sp\|or,**$g$**-bl** | $(s_i, s_j)$ | *with* $s_i > s_j$ |
| $t$**-st,ub\|nsh,co,sp\|or,**$g$**-bl** | $(s_i, s_j, s_k)$ | *with* $s_j > s_i > s_k$ |
| $t$**-st,ub\|nsh,tw,sp\|or,**$g$**-bl** | $(s_i, s_j)$ | *such that $\mathcal{I}_i$ and $\mathcal{I}_j$ overlap* |
| $t$**-st,ub\|nsh,se,**0**-sp\|or,**$g$**-bl** | $(u, v)$ | *with* $u < v$ |
| $t$**-qu,ub\|nsh,se,**0**-sp\|or,**$g$**-bl** | $(u, v)$ | *with* $u > v$ |
| $t$**-qu,ub\|nsh,co,**0**-sp\|or,**$g$**-bl** | $(u, v)$ | *with* $u > v$ |
| $t$**-qu,ub\|nsh,tw,**0**-sp\|or,**$g$**-bl** | $(u, v)$ | *with* $u > v$ |
| $t$**-st,ub\|nsh,co,**0**-sp\|or,**$g$**-bl** | $(u, v, w)$ | *with* $v > u \geq w$ |
| $t$**-st,ub\|nsh,tw,**0**-sp\|or,**$g$**-bl** | $(u = s_i, v = s_j)$ | *such that $\mathcal{I}_i$ and $\mathcal{I}_j$ overlap* |
| $t$**-st,ub\|nsh,se,**0**-sp\|fr,**$g$**-bl** | $(u, v, u)$ *or* $(v, u, v)$ | |
| $t$**-qu,ub\|nsh,se,**0**-sp\|fr,**$g$**-bl** | $(u, v, u)$ *or* $(v, u, v)$ | |
| $t$**-qu,ub\|nsh,co,**0**-sp\|fr,**$g$**-bl** | $(u, v, u)$ *or* $(v, u, v)$ | |
| $t$**-st,ub\|nsh,co,**0**-sp\|fr,**$g$**-bl** | $(u, v, u, v)$ *or* $(v, u, v, u)$ | |

*Proof.* Obvious for the three versions $t$**-st,ub\|nsh,se,sp\|or,**$g$**-bl** (row 1 of the table above), $t$**-st,ub\|nsh,tw,sp\|or,**$g$**-bl** (row 6), and $t$**-st,ub\|nsh,tw,**0**-sp\|or,**$g$**-bl** (row 12).

$t$**-st,ub\|nsh,co,sp\|or,**$g$**-bl** (row 5) : $[\Leftarrow]$ If two units $i$ and $j$ with $s_i < s_j$ that start a subsequence $(s_i, s_j, s_k)$ of $S$ with $s_k < s_i$ are stored on the same track, then the unit $i$ obviously has to have left before unit $j$ is parked. However, then in any case the unit $k$ would depart after unit $i$, which is a contradiction to the required departing order. $[\Rightarrow]$ Assume that $S$ does not contain a subsequence $(s_i, s_j, s_k)$ with $s_j > s_i > s_k$ and

we store units $i$ and $j$ with $i < j$ on the same track. Then, there are two possible cases. In case $s_i \geq s_j$, the units $i$ and $j$ can depart – $i$ later than $j$ – without causing any conflicts. In case $s_i < s_j$, unit $i$ can depart before unit $j$ is parked, since all units of group $s_k$ with $s_k < s_i$ could have left already at that time as desired.

**$t$-st,ub|nsh,se,0-sp|fr,$g$-bl** (row 13) : [$\Leftarrow$] If $S$ contains a subsequence $(u,v,u)$, the groups $u$ and $v$ cannot be stored on the same track, because a departure of a complete and absolute group $u$ would not be possible without additional rearrangements. [$\Rightarrow$] If $S$ does not contain subsequences $(u,v,u)$ and $(v,u,v)$, then the periods of $u$ and $v$ are disjoint and both groups can depart as requested when placed on the same track.

**$t$-st,ub|nsh,co,0-sp|fr,$g$-bl** (row 16) : [$\Leftarrow$] If $S$ contains a subsequence $(u,s_i = v,s_j = u,v)$ and the groups $u$ and $v$ are placed on the same track, then unit $i$ has to have left within group $v$ before unit $j$ is parked, which is not possible since another unit of group $v$ arrives afterwards. [$\Rightarrow$] On the contrary, if $S$ does not contain subsequences $(u,v,u,v)$ and $(v,u,v,u)$, the periods of $u$ and $v$ are not overlapping. If they are disjoint there are no problems in case both groups are parked on the same track. In the remaining case of containment, no unit of the other group occurs within the contained period, and the groups $u$ and $v$ located on a single track may depart as desired.

**$t$-st,ub|nsh,se,0-sp|or,$g$-bl** (row 7) : [$\Leftarrow$] Let $S$ contain a subsequence $(u,s_i = v)$ with $u < v$. If we place the groups $u$ and $v$ on the same track, then unit $i$ blocks the group $u$ at its earlier departure time. [$\Rightarrow$] If $S$ does not contain subsequences $(u,v)$ with $u < v$, then the periods of $u$ and $v$ are disjoint, in particular, $P_v \lessdot P_u$. Hence, the groups $u$ and $v$ may depart as desired in case they are parked on the same track.

**$t$-st,ub|nsh,co,0-sp|or,$g$-bl** (row 11) : [$\Leftarrow$] Let $S$ contain a subsequence $(u,s_i = v,w)$ with $u < v$ and $w \leq u$. If we store the groups $u$ and $v$ on the same track, group $u$ obviously has to have left before unit $i$ is parked. However, this is not possible because at the arrival time of unit $i$ either ($w = u$) group $u$ is not yet completed or ($w < u$) group $u$ would depart before a completed group $w$, which is a contradiction to the departing order. [$\Rightarrow$] On the contrary, if $S$ does not contain a subsequence $(u,v,w)$ with $u < v$ and $w \leq u$, then $P_u$ and $P_v$ are not overlapping and $P_u$ does not contain $P_v$. Thus, we have three remaining cases: $P_v \lessdot P_u$, $P_u \lessdot P_v$, or $P_u \subset P_v$. If $P_v \lessdot P_u$, the groups $u$ and $v$ may obviously be parked on the same track. In the other two cases, there is

no unit of group $v$ within period $P_u$. Group $u$ may depart immediately after the last unit of group $u$ and before the next unit of group $v$ is parked, since all groups $w'$ with $w' < u$ already could have left at that time as desired.

The forbidden subsequences of the **queues** versions result from those of the respective **stacks** versions, see Observation 3.2. □

Note that the pure consideration of a subsequence $(s_i, s_j)$ of $S^{\mathcal{I}}$ is not sufficient to see whether or not the units $i$ and $j$ may be stored on the same track in the **time windows** versions; we additionally need to know the intervals $\mathcal{I}_i$ and $\mathcal{I}_j$. Though, for ease of notation, we also say that $(s_i, s_j)$ is a forbidden subsequence for **time windows** versions if $\mathcal{I}_i$ and $\mathcal{I}_j$ overlap.

Remember, $t$-**minimizing** versions are equivalent to finding a minimum feasible partition of the input sequence $S$ (or of $S^{\mathcal{I}}$). Due to the knowledge of the structure of forbidden subsequences, we now are able to identify the feasibility of a partition of $S$ (or of $S^{\mathcal{I}}$) for the versions listed in Theorem 4.2, and for the respective $b$-**bounded** versions.

A partition of $S$ (or of $S^{\mathcal{I}}$) into subsequences is feasible if:

1. two units $i$ and $j$ that start a forbidden subsequence $(s_i, s_j, \ldots)$ of $S$ (or of $S^{\mathcal{I}}$) – w. r. t. the considered version – do not both belong to the same subsequence,
2. all units of one group belong to the same subsequence in case of a 0-**split** version,
3. at most $b$ units are contained in one subsequence in case of a $b$-**bounded** version.

Besides that, the **unbounded** versions $\mathcal{V}$ listed in Theorem 4.2 may be formulated as MVC of the respective $\mathcal{V}$-graphs $G^{\mathcal{V}}$, see Subsection 3.1. For these versions one can easily specify which units or groups may be placed on the same track or not. In case $\mathcal{V}$ is a 0-**split** version, two vertices (groups) $u$ and $v$ are adjacent in $G^{\mathcal{V}}$ if there are units $i$ and $j$ with $s_i = u$ and $s_j = v$ that start a forbidden subsequence $(s_i, s_j, ...)$ of $S$ (or of $S^{\mathcal{I}}$). In case $\mathcal{V}$ is a **split** version, two vertices $i$ and $j$ are adjacent if the units $i$ and $j$ start a forbidden subsequence $(s_i, s_j, ...)$ of $S$ (or of $S^{\mathcal{I}}$). Consequently, the $b$-**bounded**, **split** versions $\mathcal{V}$ which relate to **unbounded** versions listed in Theorem 4.2 are equivalent to $b$-MES of the respective $\mathcal{V}$-graphs.

### 4.3.1 Unbounded Case

In this section we classify the computational complexity of the $t$-**minimizing**, **unbounded** versions listed in Theorem 4.2 by showing their equivalence to MVC of particular graphs.

**Theorem 4.3 (Well-known for $n$-bl case)**
*Version* $\underline{t}$***-st,ub|nsh,se,sp|or,****g****-bl** *is equivalent to* MVC *of permutation graphs and can be solved in* $\mathcal{O}(n \log n)$ *time.*

*Proof.* For a given permutation $\Pi$ the respective permutation graph is obviously isomorphic to the graph $G^{\underline{t}\text{-st,ub}|\text{nsh,se,sp}|\text{or,}g\text{-bl}}(\Pi)$.

On the contrary, given a sequence $S = S_{n,g}$ we construct a permutation $\bar{\Pi}(S)$ of the numbers $1, \ldots, n$ such that $\bar{\pi}_i > \bar{\pi}_j$ if $i < j$ and $s_i \geq s_j$. For example, $\bar{\Pi}(S) = (2,3,1,6,4,5)$ for $S = (1,2,1,4,3,4)$. By definition, the graph $G^{\underline{t}\textbf{-st,ub}|\textbf{nsh,se,sp}|\textbf{or,}g\textbf{-bl}}(S)$ is isomorphic to the permutation graph $G^P(\bar{\Pi}(S))$.

Consequently, the class of $\underline{t}$**-st,ub|nsh,se,sp|or,**$g$**-bl** - graphs and the class of permutation graphs are equivalent, and the version $\underline{t}$**-st,ub|nsh,se,sp|or,**$g$**-bl** can be solved for an instance $S$ by determining an optimal coloring of the permutation graph for $\bar{\Pi}(S)$. Permutation graphs can be colored optimally with FIRST FIT – according to the order of elements – in $\mathcal{O}(n \log n)$, see EVEN & ITAI (1971). Thus, the version $\underline{t}$**-st,ub|nsh,se,sp|or,**$g$**-bl** is solvable in $\mathcal{O}(n \log n)$. □

**Implication 4.1**
*Versions* $\underline{t}$***-qu,ub|nsh,se,sp|or,****g****-bl**, $\underline{t}$***-qu,ub|nsh,co,sp|or,****g****-bl**, *and* $\underline{t}$***-qu,ub|nsh,tw,sp|or,****g****-bl** *can be solved in* $\mathcal{O}(n \log n)$ *time by determining an optimal coloring of the permutation graph* $G^P(\bar{\Pi}(S^{\updownarrow}))$.

**Theorem 4.4 (Well-known)**
*Version* $\underline{t}$***-st,ub|nsh,se,*****0*****-sp|fr,****g****-bl** *is equivalent to* MVC *of interval graphs, and it can be solved in* $\mathcal{O}(n \log n)$ *time.*

*Proof.* Remember, for each interval graph $G^I(\mathcal{I})$ w. r. t. an arbitrary interval representation $\mathcal{I}$, there exists an interval graph $G^I(\bar{\mathcal{I}})$ regarding an interval representation $\bar{\mathcal{I}}$ whose intervals all have different start and end points which is isomorphic to $G^I(\mathcal{I})$. For this graph $G^I(\bar{\mathcal{I}})$, there obviously exists a sequence $S$ such that the interval graph regarding $P(S)$ is isomorphic to $G^I(\bar{\mathcal{I}})$. Hence, each interval graph is a $\underline{t}$**-st,ub|nsh,se,0-sp|fr,**$g$**-bl** - graph.

On the contrary, two vertices $u$ and $v$ are adjacent in the graph $G^{\underline{t}\text{-}\mathbf{st,ub}|\mathbf{nsh,se,}0\text{-}\mathbf{sp}|\mathbf{fr,}g\text{-}\mathbf{bl}}(S)$ if and only if the periods $P_u$ and $P_v$ intersect. Thus, the class of $\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**fr,**$g$-**bl** - graphs equals the class of interval graphs, and the version $\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**fr,**$g$-**bl** can be solved for an instance $S$ by determining an optimal coloring of the interval graph $G^I(P(S))$. It is well-known that interval graphs with given interval representation can be colored optimally with FIRST FIT – according to the order of increasing start points – in $\mathcal{O}(n \log n)$. We obtain the interval representation $P(S)$ in $\mathcal{O}(n)$ by simply scanning the sequence $S$ from left to right. Thus, version $\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**fr,**$g$-**bl** can be solved in $\mathcal{O}(n \log n)$ time. □

**Implication 4.2**
*Versions $\underline{t}$-**qu,ub**|**nsh,se,**0-**sp**|**fr,**$g$-**bl** and $\underline{t}$-**qu,ub**|**nsh,co,**0-**sp**|**fr,**$g$-**bl** can be solved in $\mathcal{O}(n \log n)$ time by determining an optimal coloring of the interval graph $G^I(P(S^{\updownarrow}))$.*

**Theorem 4.5**
*Version $\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**or,**$g$-**bl** is equivalent to* MVC *of point-interval graphs, and it can be solved in $\mathcal{O}(n \log n)$ time.*

*Proof.* On the one hand, consider a point-interval graph $G^{PI}(\Delta)$ according to an arbitrary set $\Delta$ of $n^\Delta$ triangles. There exists a point-interval graph $G^{PI}(\bar{\Delta})$ w. r. t. a set $\bar{\Delta} = \{\bar{\Delta}_\mathfrak{g}(\mathfrak{p}_1^\mathfrak{g}, \mathfrak{p}_2^\mathfrak{g}, \mathfrak{p}_3^\mathfrak{g}) \mid \mathfrak{g} = 1, \ldots, n^\Delta\}$ of triangles such that $\left\{\mathfrak{p}_1^1, \ldots, \mathfrak{p}_1^{n^\Delta}, \mathfrak{p}_2^1, \ldots, \mathfrak{p}_2^{n^\Delta}\right\} = \left\{1, \ldots, 2 \cdot n^\Delta\right\}$ and $\left\{\mathfrak{p}_3^1, \ldots, \mathfrak{p}_3^{n^\Delta}\right\} = \{1, \ldots, n^\Delta\}$ which is isomorphic to $G^{PI}(\Delta)$. We construct a sequence $S$ of $2n^\Delta$ elements. In particular, for $\mathfrak{g} = 1, \ldots, n^\Delta$ we define $s_{\mathfrak{p}_1^\mathfrak{g}} := n^\Delta + 1 - \mathfrak{p}_3^\mathfrak{g}$ and $s_{\mathfrak{p}_2^\mathfrak{g}} := n^\Delta + 1 - \mathfrak{p}_3^\mathfrak{g}$. This sequence $S$ contains a (forbidden) subsequence $(u, v)$ with $u < v$ if and only if the triangles $\bar{\Delta}_u$ and $\bar{\Delta}_v$ intersect, see Figure 4.2. Thus, each point-interval graph is a $\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**or,**$g$-**bl** - graph.

On the other hand, given a sequence $S = S_{n,g}$, let us consider the set of triangles $\tilde{\Delta}(S) = \{\, \tilde{\Delta}_\mathfrak{g}(p_\mathfrak{g}^f(S), p_\mathfrak{g}^l(S), g + 1 - \mathfrak{g}) \mid \mathfrak{g} = 1, \ldots, g \,\}$, see Figure 4.2. By construction, the triangles $\tilde{\Delta}_u$ and $\tilde{\Delta}_v$ intersect if and only if the sequence $S$ contains a forbidden subsequence $(u, v)$ with $u < v$. That is, the graph $G^{\underline{t}\text{-}\mathbf{st,ub}|\mathbf{nsh,se,}0\text{-}\mathbf{sp}|\mathbf{or,}g\text{-}\mathbf{bl}}(S)$ is isomorphic to the point-interval graph $G^{PI}(\tilde{\Delta}(S))$.

As a consequence, the class of $\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**or,**$g$-**bl** - graphs is equivalent to the class of point-interval graphs, and the version

$\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**or,**$g$-**bl** can be solved for a given sequence $S$ by determining an optimal coloring of the point-interval graph $G^{PI}(\tilde{\Delta}(S))$. MVC of point-interval graphs can be solved in $\mathcal{O}(n \log n)$ time, see FELSNER ET AL. (1997) for an algorithm that applies to the more general class of trapezoid graphs with the same time complexity. For a rephrased version of this algorithm tailored to the terminology of point-interval graphs see Algorithm 5 below. After all, one can solve version $\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**or,**$g$-**bl** in $\mathcal{O}(n \log n)$ time. □

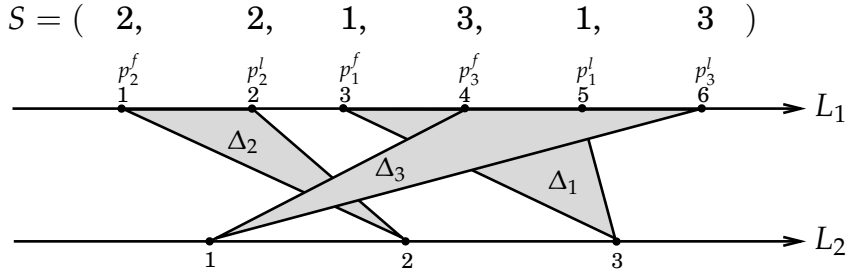


**Figure 4.2.** Relation between $\underline{t}$-**st,ub**|**nsh,se,**0-**sp**|**or,**$g$-**bl** - graphs and point-interval graphs

**Algorithm 5:** BEST FIT for MVC of point-interval graphs

**Input** : A point-interval graph $G^{PI}(\Delta)$ and its triangle representation $\Delta$ (see page 25 for the definition) whose triangles have pairwise disjoint corner points and are increasingly numbered according to the corner points $\mathfrak{p}_1^i$, that is, $\mathfrak{p}_1^1 < \mathfrak{p}_1^2 < \cdots < \mathfrak{p}_1^n$

**Output**: A minimum vertex coloring of $G^{PC}(\Delta)$ in $\mathcal{O}(n \log n)$ time

1 **for** $i \leftarrow 1$ **to** $n$ **do**

2 **If** the already used color $c^*$ exists with the properties:

- $\mathfrak{p}_2^j < \mathfrak{p}_1^i$ and $\mathfrak{p}_3^j < \mathfrak{p}_3^i$ where $\Delta_j$ with $j < i$ is the vertex colored last with color $c^*$
- there is no already used color $c$ with $\mathfrak{p}_2^l < \mathfrak{p}_1^i$ and $\mathfrak{p}_3^j < \mathfrak{p}_3^l < \mathfrak{p}_3^i$ where $\Delta_l$ with $l < i$ is the vertex colored last with color $c$

**then** color vertex $\Delta_i$ with $c^*$
**else** color vertex $\Delta_i$ with a new color

**Implication 4.3**
*Versions* $\underline{t}$***-qu,ub|nsh,se,***$0$***-sp|or,***$g$***-bl****,* $\underline{t}$***-qu,ub|nsh,tw,***$0$***-sp|or,***$g$***-bl****, and* $\underline{t}$***-qu,ub|nsh,co,***$0$***-sp|or,***$g$***-bl*** *can be solved in* $\mathcal{O}(n \log n)$ *time by determining an optimal coloring of the point-interval graph* $G^{PI}(\tilde{\Delta}(S^{\updownarrow}))$.

**Theorem 4.6 (DI STEFANO & KOČI (2004))**
*Version* $\underline{t}$***-st,ub|nsh,tw,sp|or,***$g$***-bl*** *is equivalent to* MVC *of circle graphs, and it is* $\mathcal{NP}$*-hard.*

*Proof.* For any circle graph $G^C$, there is a circle graph $G^C(\mathcal{C}(\bar{\mathcal{I}}))$ with respect to a set of chords $\mathcal{C}(\bar{\mathcal{I}})$ – the intervals in $\bar{\mathcal{I}}$ all have different start points – which is isomorphic to $G^C$, see page 26. This graph $G^C(\mathcal{C}(\bar{\mathcal{I}}))$ is isomorphic to $G^{\underline{t}\text{-}\mathbf{st,ub}|\mathbf{nsh,tw,sp}|\mathbf{or,}g\text{-}\mathbf{bl}}(\bar{\mathcal{I}})$. On the contrary, each $\underline{t}$**-st,ub|nsh,tw,sp|or,**$g$**-bl** - graph obviously is a circle graph, see Figure 2.1. Thus, version $\underline{t}$**-st,ub|nsh,tw,sp|or,**$g$**-bl** can be solved for an instance $\mathcal{I}$ by determining an optimal coloring of the circle graph $G^C(\mathcal{C}(\mathcal{I}))$. Another consequence is, that the class of $\underline{t}$**-st,ub|nsh,tw,sp|or,**$g$**-bl** - graphs equals the class of circle graphs. From the $\mathcal{NP}$-hardness of MVC of circle graphs – proven in GAREY ET AL. (1980) – it follows that the version $\underline{t}$**-st,ub|nsh,tw,sp|or,**$g$**-bl** is $\mathcal{NP}$-hard. □

**Theorem 4.7**
*Version* $\underline{t}$***-st,ub|nsh,tw,***$0$***-sp|or,***$g$***-bl*** *is equivalent to* MVC *of polygon-circle graphs, and it is* $\mathcal{NP}$*-hard.*

*Proof.* For any polygon-circle graph $G^{PC}(\mathcal{P})$, there exist reals assigned to the corner points of the polygons such that two polygons intersect if and only if the spider condition is true for the two sets of reals that correspond to the corner points of these two polygons, see Subsection 2.4.1. We construct an instance of the version $\underline{t}$**-st,ub|nsh,tw,**$0$**-sp|or,**$g$**-bl**, in other words, a set $\mathcal{I}$ of $n^c - n^{\mathcal{P}}$ intervals. In particular, let us define $\mathcal{I}^u = \bigcup_{k=1,\ldots,n_u^c-1}\{\mathcal{I}_{h(k)}\}$ with $\mathcal{I}_{h(k)} = [\mathfrak{r}_k, \mathfrak{r}_{n_u^c}]$ for $u = 1, \ldots, n^{\mathcal{P}}$, where $\mathfrak{r}_k$ is the $h(k)$-th greatest real in the set $\{\mathfrak{r}_l \in \{\mathfrak{r}_1, \ldots, \mathfrak{r}_{n^c}\} \mid \mathfrak{p}(r_l)$ is not a last corner point of a polygon$\}$. Remember, $\mathcal{I}^u$ is the set of time intervals for units of group $u$. For example, we obtain $\mathcal{I}^1 = \{\mathcal{I}_2\}$, $\mathcal{I}^2 = \{\mathcal{I}_1, \mathcal{I}_3\}$, $\mathcal{I}^3 = \{\mathcal{I}_5\}$, and $\mathcal{I}^4 = \{\mathcal{I}_4, \mathcal{I}_6\}$ with $\mathcal{I}_1 = [\mathfrak{r}_1, \mathfrak{r}_6]$, $\mathcal{I}_2 = [\mathfrak{r}_2, \mathfrak{r}_4]$, $\mathcal{I}_3 = [\mathfrak{r}_3, \mathfrak{r}_6]$, $\mathcal{I}_4 = [\mathfrak{r}_5, \mathfrak{r}_{10}]$, $\mathcal{I}_5 = [\mathfrak{r}_7, \mathfrak{r}_8]$, and $\mathcal{I}_6 = [\mathfrak{r}_9, \mathfrak{r}_{10}]$ for the set of polygons shown in Figure 2.3 on page 29. Two vertices $\mathcal{P}_u(\mathfrak{R}_u)$ and $\mathcal{P}_v(\mathfrak{R}_v)$ in $G^{PC}(\mathcal{P})$ are adjacent – that is, the spider condition is true for $\mathfrak{R}_u$ and $\mathfrak{R}_v$ – if and only if there

exist two overlapping intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ with $\mathcal{I}_i \in \mathcal{I}^u$, $\mathcal{I}_j \in \mathcal{I}^v$. Thus, each polygon-circle graph is a $\underline{t}$-**st,ub**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** - graph.

On the contrary, in each instance $\mathcal{I}$ for $\underline{t}$-**st,ub**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** a group $\mathfrak{g}$ is characterized by the arrival times of all units of this group – denoted by $A_\mathfrak{g}(\mathcal{I}) := \{ a_i \mid \mathfrak{g} = s_i \in S^\mathcal{I}, i = 1, \ldots, n \}$ – and by its departure time which we denote by $D_\mathfrak{g}(\mathcal{I})$. There exist two overlapping intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ with $\mathcal{I}_i \in \mathcal{I}^u$, $\mathcal{I}_j \in \mathcal{I}^v$ if and only if the two polygons $\mathcal{P}_u$ and $\mathcal{P}_v$ intersect in the polygon-circle graph $G^{PC}(\mathcal{P}(\mathcal{I}))$ with $\mathcal{P}(\mathcal{I}) = \{ \mathcal{P}_\mathfrak{g}(A_\mathfrak{g}(\mathcal{I}) \cup D_\mathfrak{g}(\mathcal{I})) \mid \mathfrak{g} = 1, \ldots, g \}$. In other words, version $\underline{t}$-**st,ub**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** can be solved for an instance $\mathcal{I}$ by determining an optimal coloring of the polygon-circle graph $G^{PC}(\mathcal{P}(\mathcal{I}))$. Besides that, each $\underline{t}$-**st,ub**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** - graph is a polygon-circle graph. We conclude that the class of $\underline{t}$-**st,ub**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** - graphs equals the class of polygon-circle graphs. Since the class of circle graphs is contained in the class of polygon-circle graphs, the $\mathcal{NP}$-hardness of MVC of circle graphs extends to MVC of polygon-circle graphs. As a consequence, version $\underline{t}$-**st,ub**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** is $\mathcal{NP}$-hard. □

**Theorem 4.8**
*Version* $\underline{t}$-***st,ub***|***nsh,co,***0-***sp***|***fr,***$g$-***bl*** *is equivalent to* MVC *of polygon-circle graphs, and it is* $\mathcal{NP}$*-hard.*

*Proof.* On the one hand, let a polygon-circle graph $G^{PC}(\mathcal{P})$ w. r. t. polygons $\mathcal{P}_1(\mathfrak{R}_1), \ldots, \mathcal{P}_{n^\mathcal{P}}(\mathfrak{R}_{n^\mathcal{P}})$ be given. We construct a sequence $\bar{S}$ with $n^c$ elements. In particular, for $i = 1, \ldots, n^c$ we define $\bar{s}_i := u$ if polygon $\mathcal{P}_u$ contains corner point $\mathfrak{p}(\mathfrak{r}_i)$. For example, we obtain $\bar{S} = (2, 1, 2, 1, 4, 2, 3, 3, 4, 4)$ for the polygons shown in Figure 2.3 on page 29 . The polygons $\mathcal{P}_u(\mathfrak{R}_u)$ and $\mathcal{P}_v(\mathfrak{R}_v)$ intersect in the polygon-circle graph $G^{PC}(\mathcal{P})$ if and only if $\bar{S}$ contains a forbidden subsequence $(u, v, u, v)$ or $(v, u, v, u)$. Consequently, each polygon-circle graph is a $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**fr,**$g$-**bl** - graph.

On the other hand, consider an integer sequence $S = S_{n,g}$. We define $R_\mathfrak{g} = \{i \mid \mathfrak{g} = s_i\}$ for all groups $\mathfrak{g} \in \mathcal{G}^S$. Two groups $u$ und $v$ are adjacent in the $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**fr,**$g$-**bl** - graph if and only if $S$ contains a forbidden subsequence $(u, v, u, v)$ or $(v, u, v, u)$. The sequence $S$ contains such a forbidden subsequence if and only if the spider condition is true for $\mathfrak{R}_u$ and $\mathfrak{R}_v$. The spider condition is true for $\mathfrak{R}_u$ and $\mathfrak{R}_v$ if and only if the polygons $\mathcal{P}_u(\mathfrak{R}_u)$ and $\mathcal{P}_v(\mathfrak{R}_v)$ intersect in the polygon-circle graph w. r. t. $\hat{\mathcal{P}}(S) = \{ \mathcal{P}_\mathfrak{g}(\mathfrak{R}_\mathfrak{g}) \mid \mathfrak{g} = 1, \ldots, g \}$. Thus, we

can solve version $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**fr,**$g$-**bl** for an instance $S$ by determining an optimal coloring of the polygon-circle graph $G^{PC}(\hat{\mathcal{P}}(S))$. Another consequence is, that each $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**fr,**$g$-**bl** - graph is a polygon-circle graph. After all, the class of $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**fr,**$g$-**bl** - graphs is equivalent to the class of polygon-circle graphs. Hence, version $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**fr,**$g$-**bl** is $\mathcal{NP}$-hard. □

**Theorem 4.9**
*Version* $\underline{t}$-***st,ub***|***nsh,co,***0-***sp***|***or,***$g$-***bl*** *is equivalent to* MVC *of polygon-circle graphs, and it is* $\mathcal{NP}$*-hard.*

*Proof.* Consider a polygon-circle graph $G^{PC}(\mathcal{P})$ w. r. t. polygons $\mathcal{P}_1(\mathfrak{R}_1), \ldots, \mathcal{P}_{n^{\mathcal{P}}}(\mathfrak{R}_{n^{\mathcal{P}}})$. We construct a sequence $\bar{S}$ with $n^c$ elements in the same way as in the previous proof. If the polygons $\mathcal{P}_u$ and $\mathcal{P}_v$ intersect in the polygon-circle graph $G^{PC}(\mathcal{P})$, then $\bar{S}$ contains a forbidden subsequence $(u, v, u)$. On the contrary, if $\bar{S}$ contains either a subsequence $(u, v, u)$ with $v > u$ or a subsequence $(u, v, w)$ with $v > u > w$, then it also contains a subsequence $(u, v, u, v)$ – due to the numbering of the polygons – see Convention 2.1. Hence, the polygons $\mathcal{P}_u$ and $\mathcal{P}_v$ intersect. As a consequence, each polygon-circle graph is a $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**or,**$g$-**bl** - graph.

On the other hand, let be given an integer sequence $S = S_{n,g}$, that is, an instance for version $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**or,**$g$-**bl**. We construct a set of intervals $\bar{\mathcal{I}}(S) := \{\, \mathcal{I}_i(\bar{a}_i, \bar{d}_i) \mid i = 1, \ldots, n \,\}$ where $\bar{a}_i := i$ and $\bar{d}_i := \max\{\, j \mid s_j \leq s_i,\ j \geq i \,\} + s_i \cdot \epsilon$ for an $\epsilon > 0$ sufficiently small. In other words, for each unit of $S$, we define a particular arrival and departure time in such a way that each group departs immediately after all "*preceding*" groups left. For example, the sequence $(3, 2, 3, 1, 2)$ leads to the following set of intervals: $\{[1, 5+3\epsilon], [2, 5+2\epsilon], [3, 5+3\epsilon],\ [4, 4+\epsilon], [5, 5+2\epsilon]\}$. The sequence $S$ contains a forbidden subsequence $(u, v, u)$ with $v > u$ or $(u, v, w)$ with $v > u > w$ if and only if there exist two overlapping intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ with $\mathcal{I}_i \in \mathcal{I}^u$, $\mathcal{I}_j \in \mathcal{I}^v$. Consequently, each $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**or,**$g$-**bl** - graph is a $\underline{t}$-**st,ub**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** - graph, which in proof of Theorem 4.7 is shown to be a polygon-circle graph. That is, we can solve version $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**or,**$g$-**bl** for an instance $S$ by determining an optimal coloring of the polygon-circle graph w. r. t. $\mathcal{P}(\bar{\mathcal{I}}(S))$.

After all, with the result above we conclude that the classes of $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**or,**$g$-**bl** - graphs and of polygon-circle graphs are equivalent. As a consequence, version $\underline{t}$-**st,ub**|**nsh,co,**0-**sp**|**or,**$g$-**bl** is $\mathcal{NP}$-hard. □

**Theorem 4.10**
*Version* $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl** *is equivalent to* MVC *of a proper subclass of circle graphs, and it is* $\mathcal{NP}$*-hard.*

*Proof.* Note that the graph $G = (\{u, v\}, \{\{u, v\}\})$ denoted as $K_2$ in the literature is a circle graph but no $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl**-graph.

Let be given a sequence $S = S_{n,g}$, that is, an instance for version $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl**. As for version $\underline{t}$**-st,ub|nsh,co,**$0$**-sp|or,**$g$**-bl** we define for each unit of $S$ a particular arrival and departure time. However, in this case, such that each unit departs immediately after all "*preceding*" units left, which is guaranteed by $\hat{a}_i := i$ as arrival time and $\hat{d}_i := \max\{\{\, j \mid s_j < s_i,\ j > i \,\} \cup \{i\}\} + s_i \cdot \epsilon$ for an $\epsilon > 0$ sufficiently small as departure time of unit $i$. The resulting set of intervals is denoted by $\hat{\mathcal{I}}(S) := \{\, \mathcal{I}_i(\hat{a}_i, \hat{d}_i) \mid i = 1, \ldots, n \,\}$. For example, the sequence $(3, 2, 3, 1, 2)$ leads to $\{[1, 5 + 3\epsilon], [2, 4 + 2\epsilon], [3, 5 + 3\epsilon], [4, 4 + \epsilon], [5, 5 + 2\epsilon]\}$. By above construction, the sequence $S$ contains a forbidden subsequence $(s_i, s_j, s_k)$ with $s_j > s_i > s_k$ if and only if the two intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ overlap. That is, each $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl**-graph is a $\underline{t}$**-st,ub|nsh,tw,sp|or,**$g$**-bl**-graph, which in proof of Theorem 4.6 is shown to be a circle graph. Consequently, the class of $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl**-graphs is a subclass of the class of circle graphs. Besides that, we can solve version $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl** for an instance $S$ by determining an optimal coloring of the circle graph w. r. t. $\mathcal{C}(\hat{\mathcal{I}}(S))$, that is, of the polygon-circle graph w. r. t. $\mathcal{P}(\hat{\mathcal{I}}(S))$.

Although the class of $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl**-graphs is only a subclass of the class of circle graphs, we still can show $\mathcal{NP}$-hardness of the considered version.

Let us consider graphs $\check{G}(\mathcal{I}) = (V_1 \cup V_2, E)$ where $|V_1| = |V_2| = n$, $G^C(\mathcal{C}(\mathcal{I})) = (V_1, E)$ is a circle graph, $V_2$ is an independent set in $\check{G}(\mathcal{I})$, and all start and end points of the intervals in $\mathcal{I}$ are pairwise disjoint. Clearly, an optimal coloring of $\check{G}(\mathcal{I})$ can easily be transformed into an optimal coloring of the circle graph $G^C(\mathcal{C}(\mathcal{I}))$, and vice versa. Thus, the assumption that every graph $\check{G}(\mathcal{I})$ can be colored optimally in polynomial time implies that the same holds for every circle graph, which is a contradiction, unless $\mathcal{P} = \mathcal{NP}$. Hence, MVC of the graphs $\check{G}(\mathcal{I})$ is $\mathcal{NP}$-hard.

Now, let us assume that $V_1$ contains the even and $V_2$ the odd numbers of $\{1, \ldots, 2n\}$, and that the intervals $\mathcal{I}_i = [a_i, d_i]$ in $\mathcal{I}$ are increasingly sorted by their end points, that is, $d_i < d_j$ if $i < j$. We construct another set of intervals $\check{\mathcal{I}} := \{\, \check{\mathcal{I}}_i[\check{a}_i, \check{d}_i] \mid i \in \{1, \ldots, 2n\}\}$ where

$\check{a}_i = a_i$, $\check{d}_i = d_i$ if $i$ is even and $\check{a}_i = d_{i+1} - 2\epsilon$, $\check{d}_i = d_{i+1} - \epsilon$ for an $\epsilon > 0$ sufficiently small if $i$ is odd. The graph $\check{G}(\mathcal{I})$ is isomorphic to the circle graph $G^C(\mathcal{C}(\check{\mathcal{I}}))$. The set $\{\check{a}_1, \check{a}_2, \ldots, \check{a}_{2n}\}$ of $2n$ distinct numbers is denoted by $\check{A}$. We construct an instance of version $\underline{t}$**-st,ub**|**nsh,co,sp**|**or,**$g$**-bl**, that is, we define an integer sequence $S = S_{2n,2n}$ such that $s_i = j$ if $\check{a}_j$ is the $i$-th smallest number in $\check{A}$. The sequence $S$ contains a forbidden subsequence $(s_i, s_j, s_k)$ with $s_j > s_i > s_k$ if and only if the intervals $\check{I}_{s_i}$ and $\check{I}_{s_j}$ overlap. That is, the circle graph $G^C(\mathcal{C}(\check{\mathcal{I}}))$ is isomorphic to the graph $G^{\underline{t}\text{-}\mathbf{st,ub}|\mathbf{nsh,co,sp}|\mathbf{or,}g\text{-}\mathbf{bl}}(S)$. As a consequence, the class of the graphs $\check{G}(\mathcal{I})$ is a subclass of the class of $\underline{t}$**-st,ub**|**nsh,co,sp**|**or,**$g$**-bl** - graphs. Due to the fact that MVC of the graphs $\check{G}(\mathcal{I})$ is $\mathcal{NP}$-hard, MVC of $\underline{t}$**-st,ub**|**nsh,co,sp**|**or,**$g$**-bl** - graphs is also $\mathcal{NP}$-hard. □

Table 4.1 summarizes the main results presented in this subsection regarding the computational complexity of the listed **unbounded** versions. These results answer the question stated in Section 3.2 on page 38. That is, there are particular **queues** versions which are "*easily*", i. e., polynomially, solvable whose corresponding **stacks** versions are "*hard*" to solve, i. e., the existence of a polynomial time algorithm for the **stacks** version is unlikely. For example, the version $\underline{t}$**-qu,ub**|**nsh,co,**0**-sp**|**fr,**$g$**-bl** is solvable in $\mathcal{O}(n \log n)$ time, whereas the version $\underline{t}$**-st,ub**|**nsh,co,**0**-sp**|**fr,**$g$**-bl** is $\mathcal{NP}$-hard.

After all, among the versions $\underline{t}$**-st,ub**|**nsh,**$\cdot$**,**{**sp,**0**-sp**}|{**fr,or**}**,**$g$**-bl** and $\underline{t}$**-qu,ub**|**nsh,**$\cdot$**,**{**sp,**0**-sp**}|{**fr,or**}$g$**-bl**, the computational complexity remains open for the equivalent versions

- $\underline{t}$**-st,ub**|**nsh,se,sp**|**fr,**$g$**-bl**,
- $\underline{t}$**-qu,ub**|**nsh,se,sp**|**fr,**$g$**-bl**,
- $\underline{t}$**-qu,ub**|**nsh,co,sp**|**fr,**$g$**-bl**,

as well as for version

- $\underline{t}$**-st,ub**|**nsh,co,sp**|**fr,**$g$**-bl**.

| Version | Equivalence to MVC of | Comput. Complexity |
|---|---|---|
| $\underline{t}$**-st,ub**\|**nsh,se,sp**\|**or,**$g$**-bl** | permutation graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-qu,ub**\|**nsh,se,sp**\|**or,**$g$**-bl** | permutation graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-qu,ub**\|**nsh,co,sp**\|**or,**$g$**-bl** | permutation graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-qu,ub**\|**nsh,tw,sp**\|**or,**$g$**-bl** | permutation graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-st,ub**\|**nsh,se,**0**-sp**\|**fr,**$g$**-bl** | interval graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-qu,ub**\|**nsh,se,**0**-sp**\|**fr,**$g$**-bl** | interval graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-qu,ub**\|**nsh,co,**0**-sp**\|**fr,**$g$**-bl** | interval graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-st,ub**\|**nsh,se,**0**-sp**\|**or,**$g$**-bl** | point-interval graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-qu,ub**\|**nsh,se,**0**-sp**\|**or,**$g$**-bl** | point-interval graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-qu,ub**\|**nsh,co,**0**-sp**\|**or,**$g$**-bl** | point-interval graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-qu,ub**\|**nsh,tw,**0**-sp**\|**or,**$g$**-bl** | point-interval graphs | $\mathcal{O}(n \log n)$ |
| $\underline{t}$**-st,ub**\|**nsh,co,sp**\|**or,**$g$**-bl** | subclass of circle gr. | $\mathcal{NP}$-hard |
| $\underline{t}$**-st,ub**\|**nsh,tw,sp**\|**or,**$g$**-bl** | circle graphs | $\mathcal{NP}$-hard |
| $\underline{t}$**-st,ub**\|**nsh,co,**0**-sp**\|**or,**$g$**-bl** | polygon-circle graphs | $\mathcal{NP}$-hard |
| $\underline{t}$**-st,ub**\|**nsh,tw,**0**-sp**\|**or,**$g$**-bl** | polygon-circle graphs | $\mathcal{NP}$-hard |
| $\underline{t}$**-st,ub**\|**nsh,co,**0**-sp**\|**fr,**$g$**-bl** | polygon-circle graphs | $\mathcal{NP}$-hard |

**Table 4.1.** Computational complexity of some $t$-**minimizing**, **unbounded**, **no shunting**, $g$-**blocks** versions

### 4.3.2 Bounded Case

In the following we classify the computational complexity for the $b$-**bounded** versions that correspond to the $t$-**minimizing**, **unbounded** versions dealt with in the previous subsection by exploiting their connection to $b$-MES.

#### $t$-minimizing, $b$-bounded, split versions

Remember, if a $t$-**minimizing**, **unbounded**, **split** version is equivalent to MVC of particular graphs, then the respective $b$-**bounded** version is equivalent to $b$-MES of these graphs, see Subsection 3.1.

In the previous subsection some $t$-**minimizing**, **unbounded**, **split** versions are shown to be equivalent to MVC of permutation graphs. In JANSEN (2003) it is proven that $b$-MES of permutation graphs is $\mathcal{NP}$-hard, which is true even for fixed $b \geq 6$. As a consequence, the versions $\underline{t}$-**st,**$b$-**bd**|**nsh,se,sp**|**or,**$g$-**bl**, $\underline{t}$-**qu,**$b$-**bd**|**nsh,se,sp**|**or,**$g$-**bl**, $\underline{t}$-**qu,**$b$-**bd**|**nsh,co,sp**|**or,**$g$-**bl**, as well as $\underline{t}$-**qu,**$b$-**bd**|**nsh,tw,sp**|**or,**$g$-**bl** are $\mathcal{NP}$-hard for fixed $b \geq 6$.

The version $\underline{t}$-**st,ub**|**nsh,tw,sp**|**or,**$g$-**bl** is equivalent to MVC of circle graphs, see Subsection 4.3.1. Since the class of permutation graphs is contained in the class of circle graphs, the $\mathcal{NP}$-hardness of $b$-MES of permutation graphs for fixed $b \geq 6$ extends to circle graphs. Thus, version $\underline{t}$-**st,**$b$-**bd**|**nsh,tw,sp**|**or,**$g$-**bl** is $\mathcal{NP}$-hard for fixed $b \geq 6$.

#### $t$-minimizing, $b$-bounded, 0-split versions

Note, if a $t$-**minimizing**, **unbounded**, 0-**split** version is equivalent to MVC of particular graphs, then the respective $b$-**bounded** version is a specialization of BPC regarding these (conflict) graphs, see Subsection 3.1.

The version $\underline{t}$-**st,**$b$-**bd**|**nsh,tw,**0-**sp**|**or,**$n$-**bl** – in which exactly one unit of each group arrives – is also equivalent to $b$-MES of circle graphs, see the proof of Theorem 4.7. As a consequence, the version $\underline{t}$-**st,**$b$-**bd**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** is $\mathcal{NP}$-hard for fixed $b \geq 6$.

Besides that, the version $\underline{t}$-**st,**$b$-**bd**|**nsh,se,**0-**sp**|**or,**$n$-**bl** is equivalent to $b$-MES of permutation graphs, which is easily seen by previous results. As a consequence, versions $\underline{t}$-**st,**$b$-**bd**|**nsh,se,**0-**sp**|**or,**$g$-**bl**, $\underline{t}$-**qu,**$b$-**bd**|**nsh,se,**0-**sp**|**or,**$g$-**bl**, $\underline{t}$-**qu,**$b$-**bd**|**nsh,co,**0-**sp**|**or,**$g$-**bl**, as well as $\underline{t}$-**qu,**$b$-**bd**|**nsh,tw,**0-**sp**|**or,**$g$-**bl** are $\mathcal{NP}$-hard for fixed $b \geq 6$.

**Theorem 4.11**
*Versions* $\underline{t}$**-st,**$b$**-bd**|**nsh,se,**$0$**-sp**|**fr,**$g$**-bl**, $\underline{t}$**-qu,**$b$**-bd**|**nsh,se,**$0$**-sp**|**fr,**$g$**-bl**, *and* $\underline{t}$**-qu,**$b$**-bd**|**nsh,co,**$0$**-sp**|**fr,**$g$**-bl** *are* $\mathcal{NP}$*-hard for fixed* $b \geq 8$ *and versions* $\underline{t}$**-st,**$b$**-bd**|**nsh,co,**$0$**-sp**|**or,**$g$**-bl**, $\underline{t}$**-st,**$b$**-bd**|**nsh,co,**$0$**-sp**|**fr,**$g$**-bl** *are* $\mathcal{NP}$*-hard for fixed* $b \geq 12$.

*Proof.* By $\mathcal{V}^*$ we denote one of the five considered $b$**-bounded** versions $\mathcal{V}$ restricted to input sequences containing exactly two railcars of each group. $\tilde{\mathcal{V}}_{b\text{-bd}}$ refers to any of these versions $\mathcal{V}^*$ for short. Obviously, $\tilde{\mathcal{V}}_{b\text{-bd}}$ is equivalent to $b/2$-MES of the respective $\tilde{\mathcal{V}}_{b\text{-bd}}$ - graph. Besides that, $\tilde{\mathcal{V}}_{b\text{-bd}}$ is equivalent to $\tilde{\mathcal{V}}_{b+1\text{-bd}}$ for even $b$.

The class of $\underline{t}$**-st,**$b$**-bd**|**nsh,se,**$0$**-sp**|**fr,**$g$**-bl**$^*$ - graphs, the class of $\underline{t}$**-qu,**$b$**-bd**|**nsh,se,**$0$**-sp**|**fr,**$g$**-bl**$^*$ - graphs, as well as the class of $\underline{t}$**-qu,**$b$**-bd**|**nsh,co,**$0$**-sp**|**fr,**$g$**-bl**$^*$ - graphs equal the class of interval graphs, see the proof of Theorem 4.4 and Implication 4.2. Because $b$-MES of interval graphs is $\mathcal{NP}$-hard for fixed $b \geq 4$ – see BODLAENDER & JANSEN (1995) – the versions $\underline{t}$**-st,**$b$**-bd**|**nsh,se,**$0$**-sp**|**fr,**$g$**-bl**, $\underline{t}$**-qu,**$b$**-bd**|**nsh,se,**$0$**-sp**|**fr,**$g$**-bl**, as well as $\underline{t}$**-qu,**$b$**-bd**|**nsh,co,**$0$**-sp**|**fr,**$g$**-bl** are $\mathcal{NP}$-hard for fixed $b \geq 8$.

Finally, the class of $\underline{t}$**-st,**$b$**-bd**|**nsh,co,**$0$**-sp**|**or,**$g$**-bl**$^*$ - graphs is equivalent to the class of circle graphs, see the proof of Theorem 4.9, and the class of $\underline{t}$**-st,**$b$**-bd**|**nsh,co,**$0$**-sp**|**fr,**$g$**-bl**$^*$ - graphs contains the class of circle graphs, see the proof of Theorem 4.8. Due to the $\mathcal{NP}$-hardness of $b$-MES of circle graphs for fixed $b \geq 6$, it follows that the versions $\underline{t}$**-st,**$b$**-bd**|**nsh,co,**$0$**-sp**|**or,**$g$**-bl** and $\underline{t}$**-st,**$b$**-bd**|**nsh,co,**$0$**-sp**|**fr,**$g$**-bl** are $\mathcal{NP}$-hard for fixed $b \geq 12$. □

Table 4.2 summarizes the main results presented in this subsection regarding the computational complexity of the listed $b$**-bounded** versions.

Note that the **split** versions listed in Table 4.2 are polynomially solvable for $b = 2$, see Section 3.1. For any of these versions, it is a theoretically interesting open problem to identify the number $b$ for which the $b$-bounded case is polynomially solvable and the $b+1$-bounded case is $\mathcal{NP}$-hard. However, in most rail yards more than 12 units can be parked on the sorting tracks, in other words, it is $b > 12$ for most $b$**-bounded** real-world versions. That is, the results above indicate that for applications relating to the listed versions one might have to put some effort – depending on the dimension and structure of the actual input data – into computing an optimal solution.

| Version | problem | conflict graph | $\mathcal{NP}$-hard for | approx. ratio |
|---|---|---|---|---|
| $\underline{t}$**-st,**$b$**-bd\|nsh,se,sp\|or,**$g$**-bl** | $b$-MES | permutation | fixed $b \geq 6$ | 2.5 |
| $\underline{t}$**-qu,**$b$**-bd\|nsh,se,sp\|or,**$g$**-bl** | | | fixed $b \geq 6$ | 2.5 |
| $\underline{t}$**-qu,**$b$**-bd\|nsh,co,sp\|or,**$g$**-bl** | | | fixed $b \geq 6$ | 2.5 |
| $\underline{t}$**-qu,**$b$**-bd\|nsh,tw,sp\|or,**$g$**-bl** | | | fixed $b \geq 6$ | 2.5 |
| $\underline{t}$**-st,**$b$**-bd\|nsh,se,**0**-sp\|fr,**$g$**-bl** | BPC | interval | fixed $b \geq 8$ | $7/3$ |
| $\underline{t}$**-qu,**$b$**-bd\|nsh,se,**0**-sp\|fr,**$g$**-bl** | | | fixed $b \geq 8$ | $7/3$ |
| $\underline{t}$**-qu,**$b$**-bd\|nsh,co,**0**-sp\|fr,**$g$**-bl** | | | fixed $b \geq 8$ | $7/3$ |
| $\underline{t}$**-st,**$b$**-bd\|nsh,se,**0**-sp\|or,**$g$**-bl** | BPC | point- interval | fixed $b \geq 6$ | 2.5 |
| $\underline{t}$**-qu,**$b$**-bd\|nsh,se,**0**-sp\|or,**$g$**-bl** | | | fixed $b \geq 6$ | 2.5 |
| $\underline{t}$**-qu,**$b$**-bd\|nsh,co,**0**-sp\|or,**$g$**-bl** | | | fixed $b \geq 6$ | 2.5 |
| $\underline{t}$**-qu,**$b$**-bd\|nsh,tw,**0**-sp\|or,**$g$**-bl** | | | fixed $b \geq 6$ | 2.5 |
| $\underline{t}$**-st,**$b$**-bd\|nsh,co,sp\|or,**$g$**-bl** | $b$-MES | subclass of circle | general $b$ | $\mathcal{H}_{\tilde{b}}$ |
| $\underline{t}$**-st,**$b$**-bd\|nsh,tw,sp\|or,**$g$**-bl** | $b$-MES | circle | fixed $b \geq 6$ | $\mathcal{H}_{\tilde{b}}$ |
| $\underline{t}$**-st,**$b$**-bd\|nsh,co,**0**-sp\|or,**$g$**-bl** | BPC | polygon-circle | fixed $b \geq 12$ | |
| $\underline{t}$**-st,**$b$**-bd\|nsh,tw,**0**-sp\|or,**$g$**-bl** | | | fixed $b \geq 6$ | |
| $\underline{t}$**-st,**$b$**-bd\|nsh,co,**0**-sp\|fr,**$g$**-bl** | | | fixed $b \geq 12$ | |

**Table 4.2.** Computational complexity of some $t$**-minimizing**, $b$**-bounded**, **no shunting**, $g$**-blocks** versions

We conclude this section by applying known approximability results to the considered $b$-**bounded** versions. The results are contained in the last column of Table 4.2.

The 0-**split** versions listed in Table 4.2 cannot be approximated in polynomial time within a factor smaller than $3/2$, unless $\mathcal{P} = \mathcal{NP}$, see Section 3.1.

EPSTEIN & LEVIN (2008) present a 2.5-approximation for BPC on perfect graphs – which contain permutation graphs and point-interval graphs – as well as a $7/3$-approximation for BPC on interval graphs. Due to previous results, these approximations can also be applied to some $b$-**bounded** versions listed in Table 4.2. Remember that for any given class of (conflict) graphs, BPC generalizes $b$-MES for any fixed $b$.

Consider the following well-known greedy method MAXIS for approximating $b$-MES.

**Algorithm 6:** MAXIS for $b$-MES

**Input** : A graph $G = (V, E)$ and an integer $b$
**Output**: A feasible solution for $b$-MES of graph $G$

```
1 color ← 1, V̄ ← V
2 while V̄ ≠ ∅ do
3 |   Compute an MIS I_max = {v_1, ..., v_r} in V̄
4 |   if r > b then I_max ← {v_1, ..., v_b}
5 |   Color I_max with color, color ← color +1, V̄ ← V̄\I_max
```

For graphs $G$, where one can compute an MIS of every induced subgraph of $G$ in polynomial time, MAXIS is an approximation for $b$-MES with performance guaranty $\mathcal{H}_{\tilde{b}}$, where $\tilde{b} := \min\{b, \alpha(G)\}$ and $\mathcal{H}_{\tilde{b}}$ is the $\tilde{b}$-th harmonic number $\sum_{i=1}^{\tilde{b}} \frac{1}{i}$, see the greedy method for SET COVERING introduced by JOHNSON (1973). Since MIS is solvable in polynomial time for circle graphs, see Subsection 2.4.1, MAXIS is an $\mathcal{H}_{\tilde{b}}$-approximation for the versions $\underline{t}$-**st,**$b$-**bd**|**nsh,co,sp**|**or,**$g$-**bl** and $\underline{t}$-**st,**$b$-**bd**|**nsh,tw,sp**|**or,**$g$-**bl**.

## 4.4 Versions applied at Hump Yards

The versions dealt with in this section relate to the particular application which was the focus of a research project together with BASF in Ludwigshafen. Part of the results for these versions were already published in HANSMANN & ZIMMERMANN (2008). Let us start with a brief description of the application introducing the corresponding versions of SRS.

**Figure 4.3.** Railcars being pushed over the hump (left) such that they roll on appropriately chosen tracks (right). Pictures of the hump yard at the site of BASF, The Chemical Company, Ludwigshafen

Internal logistics at the site of BASF in Ludwigshafen is mainly based on rail transport. The aim of the project was to provide our practical partner with effective schedules for rearranging trains. Though there are a few rail yards at the site, nearly all rearrangements are performed using the main hump yard, see Figure 4.3.

For each incoming railcar, we know with which outbound train and in which block within the train it has to leave. Each outbound train serves several factory buildings and the railcars requested by one fac-

tory form a block. To assure a secure and efficient handling, the outbound trains consist in suitably **ordered blocks** matching the served buildings. Hence, the railcars of the respective block can simply be decoupled at the rear of the train, which prevents shunting in the streets.

The outbound trains are formed on parallel output tracks. Note, the number of trains to form at a time usually does not exceed the number of available parallel output tracks. In the notation of SRS, above requirements relate to the $o$-**ordered** $g$-**blocks** case. Remember, in $o$-**ordered** versions the input information can easily be encoded into one integer sequence. For example, the practical input data, that is, the input sequence with the information about (train numbers, block numbers) in $((2,1),(1,3),(1,2),(2,2),(1,1),(2,2),(1,3),(1,1))$ – i. e., the first incoming railcar has to leave in the first block of outbound train 2 – is encoded as $(4,3,2,5,1,5,3,1)$ with $g_1 = 3$ and $g_2 = 2$, see page 33.

The $h$-**hump-shunting** case – already described in Section 1.2 – is tailored to the BASF application. That is, the movement of railcars proceeds as follows: At first the entire input sequence of railcars is pushed over the hump such that the railcars roll either to an output track (i-o-move) or to a sorting track (t-t-move). Afterwards, at each humping step, all railcars placed on one track are pulled back over the hump and then each of these railcars is again pushed over the hump rolling either to an output-track (t-o-move) or to a sorting track (t-t-move).

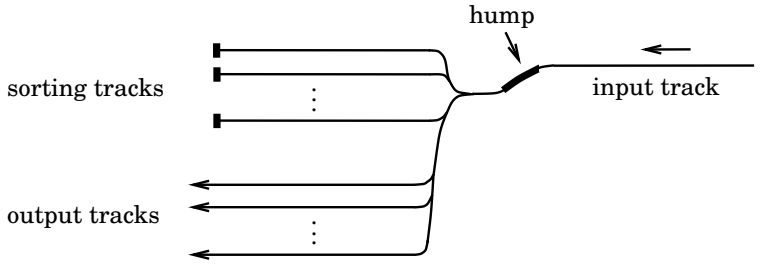


**Figure 4.4.** Topology of hump yard in case of a **stacks**, $h$-**hump-shunting**, $o$-**ordered** version

The sorting tracks in the hump yard behave like **stacks**, see Figure 4.4 to get a quick impression of the topology of the hump yard in case of a **stacks**, $h$-**hump-shunting**, $o$-**ordered** version. Note that i-o-moves are allowed, that is, arrival and departure are **concurrent**, and railcars of a group may arbitrarily **split** up over sorting tracks.

The overall goal is to form the outbound trains as quick as possible. Our practical partner confirmed, that the total amount of time spent for the rearrangements mainly depends on the number of humping steps. Thus, the objective with highest priority is to realize the sorting with minimum number $h^*$ of humping steps for a given number of free sorting tracks. The respective problem of determining a schedule with $h^*$ humping steps corresponds to version $t$-**st,**$b$-**bd**|$\underline{h}$-**hsh,co,sp**|$o$-**or,**$g$-**bl**. However, the duration of performing the rearrangements also increases with the number $r$ of *railcar moves*. This number adds up the number of t-t-moves and t-o-moves of the railcars over all humping steps, see page 9 for the definition of t-t-moves and t-o-moves. For example, $r = 12$ for the schedule shown in Figure 4.5. In order to provide our practical partner with solutions of the desired quality, we have to determine a schedule with minimum number $r^*$ of railcar moves given the minimum number $h^*$ of humping steps. We denote this real-world optimization problem by $t$-**st,**$b$-**bd**|$\underline{h}, \underline{r}$-**hsh,co,sp**|$o$-**or,**$g$-**bl**. Computational results for practical data are presented in Section 6.2.

In this section we investigate the theoretical complexity of versions related to above version matching the requirements of BASF. First of all, we consider the respective **ordered** versions for forming only one outbound train. In particular, we develop a computationally very fast algorithm for solving the versions $\underline{t}$-**st,ub**|$h$-**hsh,co,sp**|**or,**$g$-**bl** and $t$-**st,ub**|$\underline{h}$-**hsh,co,sp**|**or,**$g$-**bl**. We previously presented this algorithm at OR 2006, Karlsruhe, September 2006. In JACOB ET AL. (2007) seemingly the same algorithm is described in different terminology for the special case that all units of the input sequence are distinct.

For bookkeeping purposes, we number the tracks, and we consider the *track execution order* $\mathcal{E} = (e_1, \ldots, e_h)$. At humping step $i$, the $e_i$-th track is *executed*. For each unit, we consider its *path* through the tracks, i. e., the integer sequence of the track numbers of visited tracks. In particular, the path of a unit moving directly from the input to the output corresponds to the empty sequence (), denoted by $\emptyset$. Two easily derived observations – cf. the example shown in Figure 4.5 – are:

**Observation 4.1**
*Units taking the same path correspond to a subsequence of the input sequence.*

**Observation 4.2**
*A path corresponds to a subsequence of the track execution order.*

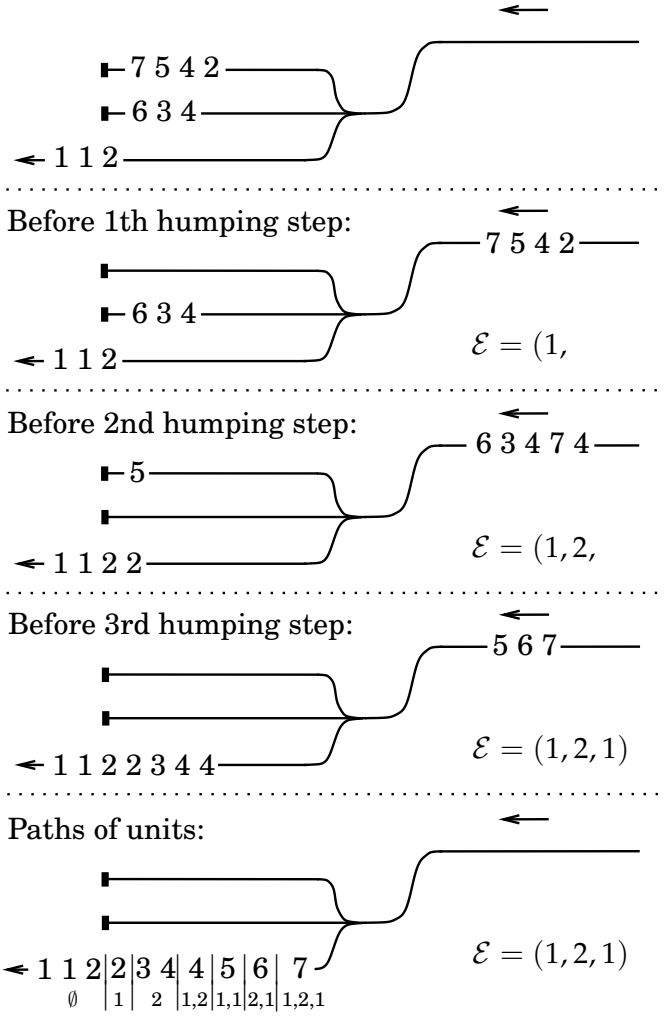


**Figure 4.5.** Optimal schedule, shunting moves over the hump, as well as the paths of the units for the instance $(7,6,5,4,3,4,1,2,1,2)$ for $\underline{t}$**-st,ub**|3**-hsh,co,sp**|**or,**$g$**-bl** and 2**-st,ub**|$\underline{h}$**-hsh,co,sp**|**or,**$g$**-bl**

For fixed number of tracks and humping steps, the number of different paths – along which a unit can move – depends on the choice of the track execution order. We call such a path realizable. For example, for two tracks and three humping steps, we consider the two track execution orders $\mathcal{E}_1 = (1,2,2)$ and $\mathcal{E}_2 = (1,2,1)$. As shown in Figure 4.5, $\mathcal{E}_2$ admits seven different realizable paths; on the other hand, $\mathcal{E}_1$ only admits the six paths $\emptyset$, $(1)$, $(2)$, $(1,2)$, $(2,2)$, $(1,2,2)$. The following corollary contains a recursion for the maximum number of different re-

alizable paths and shows that it is achieved for so-called cyclic track execution.

**Corollary 4.3**
*For $t \geq 1$ tracks in the rail yard and $h \geq 0$ humping steps the maximal number $f(h,t)$ of different realizable paths is achieved by performing the humping steps according to the **cyclic track execution order** $\mathcal{E}^c = (1,2,\ldots,t,1,2,\ldots)$ of length $h$. A recursion for $f(h,t)$ is $f(h+1,t) = 2 \cdot f(h,t) - f(h-t,t)$ for $h \geq t$ with starting values $f(h,t) = 2^h$ for $h \leq t$.*

Corollary 4.3 is an immediate consequence of Observation 4.2 and of the observations made in the proof of the following theorem.

**Theorem 4.12**
*Among all integer sequences $S_{n,g}$ of length $n$ containing integers from $\{1,\ldots,g\}$ the **cyclic sequence** $S^c_{n,g} = (1,2,\ldots,g,1,2,\ldots)$ contains the maximum number of different subsequences.*

*Proof.* We denote the number of different subsequences of an arbitrary sequence $S$ by $\bar{f}(S)$, the number of different subsequences of $S$ ending with integer $\mathfrak{g}$ by $\bar{f}_{\mathfrak{g}}(S)$, and the position of the rightmost integer $\mathfrak{g}$ in $S$ by $\tau(S,\mathfrak{g})$. It suffices to show, that $\bar{f}(S_{n,g}) \leq \bar{f}(S^c_{n,g})$ for all subsequences $S_{n,g}$ with $n > g$ containing integers $1,\ldots,g$ only.

For an inductive proof, let us define $S_{0,g} := \emptyset$ and $S_{i,g} := (s_1,\ldots,s_i)$ for $i = 1,\ldots,n-1$. We show that

$$\textcircled{1} \qquad \bar{f}\big(S^c_{i,g}\big) \;\geq\; \bar{f}\big(S_{i,g}\big)$$

for some $i$ with $g \leq i < n$ implies the corresponding inequality for $i+1$. We observe the following properties of $S_{i,g}$:

$$\begin{aligned}
\textcircled{2} \qquad \bar{f}\big(S_{i+1,g}\big) &= 2 \cdot \bar{f}\big(S_{i,g}\big) - \bar{f}_{s_{i+1}}\big(S_{i,g}\big),\\
\textcircled{3} \qquad \bar{f}\big(S_{i,g}\big) &= 1 + \textstyle\sum_{\mathfrak{g}=1}^{g} \bar{f}_{\mathfrak{g}}\big(S_{i,g}\big),\\
\textcircled{4} \qquad \bar{f}_{\mathfrak{g}}\big(S_{i,g}\big) &= \bar{f}\big(S_{\tau(S_{i,g},\mathfrak{g})-1,g}\big), \quad \mathfrak{g} = 1,\ldots,g.
\end{aligned}$$

For an easier comparison with $S^c_{i+1,g}$, we renumber the integers in $S_{i+1,g}$ and we denote the result by $\tilde{S}_{i+1,g} := (\tilde{s}_1,\ldots,\tilde{s}_{i+1})$ such that

$$\textcircled{5} \qquad \tau(\tilde{S}_{i,g},\mathfrak{g}) \;\leq\; \tau(S^c_{i,g},\mathfrak{g}), \quad \mathfrak{g} = 1,\ldots,g.$$

For example, $\tilde{S}_{6,3} = (3,3,1,2,2,1)$ for $S_{6,3} = (2,2,3,1,1,3)$, due to $S^c_{5,3} = (1,2,3,1,2)$. Of course, it holds

$$⑥ \qquad \bar{f}(\tilde{S}_{j,g}) \;=\; \bar{f}(S_{j,g}), \quad j = 1,\ldots,i+1.$$

Another consequence is that $\tau(\tilde{S}_{i,g}, s^c_{i+1}) \leq \tau(\tilde{S}_{i,g}, \tilde{s}_{i+1})$, which leads to $\bar{f}\big(\tilde{S}_{\tau(\tilde{S}_{i,g},s^c_{i+1})-1,g}\big) \leq \bar{f}\big(\tilde{S}_{\tau(\tilde{S}_{i,g},\tilde{s}_{i+1})-1,g}\big)$ and with ④ to

$$⑦ \qquad \bar{f}_{s^c_{i+1}}(\tilde{S}_{i,g}) \;\leq\; \bar{f}_{\tilde{s}_{i+1}}(\tilde{S}_{i,g}).$$

Finally, the following inequalities complete the proof.

$$\begin{aligned}
\bar{f}(S_{i+1,g}) &\overset{⑥}{=} \bar{f}(\tilde{S}_{i+1,g}) \overset{②}{=} 2\cdot\bar{f}(\tilde{S}_{i,g}) - \bar{f}_{\tilde{s}_{i+1}}(\tilde{S}_{i,g}) \\
&\overset{⑦}{\leq} 2\cdot\bar{f}(\tilde{S}_{i,g}) - \bar{f}_{s^c_{i+1}}(\tilde{S}_{i,g}) \\
&\overset{③}{=} 2\cdot\Big(1+\sum_{\mathfrak{g}=1}^{g}\bar{f}_{\mathfrak{g}}(\tilde{S}_{i,g})\Big) - \bar{f}_{s^c_{i+1}}(\tilde{S}_{i,g}) \\
&\overset{④}{=} 2+2\sum_{\substack{\mathfrak{g}=1\\ \mathfrak{g}\neq s^c_{i+1}}}^{g}\bar{f}\big(\tilde{S}_{\tau(\tilde{S}_{i,g},\mathfrak{g})-1,g}\big) + \bar{f}\big(\tilde{S}_{\tau(\tilde{S}_{i,g},s^c_{i+1})-1,g}\big) \\
&\overset{⑤}{\leq} 2+2\sum_{\substack{\mathfrak{g}=1\\ \mathfrak{g}\neq s^c_{i+1}}}^{g}\bar{f}\big(\tilde{S}_{\tau(S^c_{i,g},\mathfrak{g})-1,g}\big) + \bar{f}\big(\tilde{S}_{\tau(S^c_{i,g},s^c_{i+1})-1,g}\big) \\
&\overset{⑥}{=} 2+2\sum_{\substack{\mathfrak{g}=1\\ \mathfrak{g}\neq s^c_{i+1}}}^{g}\bar{f}\big(S_{\tau(S^c_{i,g},\mathfrak{g})-1,g}\big) + \bar{f}\big(S_{\tau(S^c_{i,g},s^c_{i+1})-1,g}\big) \\
&\overset{①}{\leq} 2+2\sum_{\substack{\mathfrak{g}=1\\ \mathfrak{g}\neq s^c_{i+1}}}^{g}\bar{f}\big(S^c_{\tau(S^c_{i,g},\mathfrak{g})-1,g}\big) + \bar{f}\big(S^c_{\tau(S^c_{i,g},s^c_{i+1})-1,g}\big) \\
&\overset{④}{=} 2\cdot\Big(1+\sum_{\mathfrak{g}=1}^{g}\bar{f}_{\mathfrak{g}}(S^c_{i,g})\Big) - \bar{f}_{s^c_{i+1}}(S^c_{i,g}) \\
&\overset{③}{=} 2\cdot\bar{f}(S^c_{i,g}) - \bar{f}_{s^c_{i+1}}(S^c_{i,g}) \overset{②}{=} \bar{f}(S^c_{i+1,g})
\end{aligned}$$ □

HIRSCHBERG & RÉGNIER (2000) prove that among all integer sequences $S_{n,g}$ of length $n$ containing integers from $\{1,\ldots,g\}$ the cyclic sequence $S^c_{n,g}$ contains the maximum number of different subsequences with cardinality $n-\mathfrak{c}$ for $\mathfrak{c} = 0,\ldots,n$. Clearly, this implies the above Theorem 4.12. They also provide recursions for computing each of these

maximum numbers in $\mathcal{O}(\mathfrak{c} + (g-1)n^2)$ time. In the above direct proof with respect to subsequences of arbitrary cardinality, we obtain a recursion for the total number $\bar{f}(S^c_{n,g})$ of all different subsequences of $S^c_{n,g}$, and we derive an explicit formula for this maximum number, cf. Corollary 4.4. Applying our recursion, we can compute $f(n,g) = \bar{f}(S^c_{n,g})$ significantly faster in $\mathcal{O}(n)$ time.

**Corollary 4.4**
*For the maximal number $f(h,t)$ of different realizable paths an explicit formula reads*

$$f(h,t) = 2^h + \sum_{j=1}^{\left\lfloor \frac{h}{t+1} \right\rfloor} (-1)^j \cdot \binom{h - j \cdot t}{j} \cdot 2^{h-j(t+1)}.$$

*Proof.* Obviously, the explicit formula yields the starting values of above recursion for $h \leq t$. It remains to show that it computes the same values as the recursion for $h > t$. By basic calculus we get

$$\begin{aligned}
&① \quad \binom{h-j\cdot t}{j} + \binom{h-j\cdot t}{j-1} = \binom{h+1-j\cdot t}{j}, \quad 1 \leq j \leq \left\lfloor \tfrac{h+1}{t+1} \right\rfloor, \\
&② \quad \binom{h+1-\frac{h+1}{t+1}\cdot t}{\frac{h+1}{t+1}} = \binom{h-\frac{h+1}{t+1}\cdot t}{\frac{h+1}{t+1}-1} = 1, \qquad \text{if } \tfrac{h+1}{t+1} \in \mathbb{Z}_+ \\
&③ \quad \left\lfloor \tfrac{h-t}{t+1} \right\rfloor + 1 = \left\lfloor \tfrac{h+1}{t+1} \right\rfloor \geq \left\lfloor \tfrac{h}{t+1} \right\rfloor.
\end{aligned}$$

Of course, it is either $\left\lfloor \frac{h+1}{t+1} \right\rfloor = \left\lfloor \frac{h}{t+1} \right\rfloor$ or $\left\lfloor \frac{h+1}{t+1} \right\rfloor = \frac{h+1}{t+1} = \left\lfloor \frac{h}{t+1} \right\rfloor + 1$.

In case of $\left\lfloor \frac{h+1}{t+1} \right\rfloor = \left\lfloor \frac{h}{t+1} \right\rfloor$,

$$\begin{aligned}
f(h+1,t) &= 2^{h+1} + \sum_{j=1}^{\left\lfloor \frac{h+1}{t+1} \right\rfloor} (-1)^j \cdot \binom{h+1-j\cdot t}{j} \cdot 2^{h+1-j(t+1)} \\
&\overset{①}{=} 2^{h+1} \\
&\qquad + \sum_{j=1}^{\left\lfloor \frac{h+1}{t+1} \right\rfloor} (-1)^j \cdot \left( \binom{h-j\cdot t}{j} + \binom{h-j\cdot t}{j-1} \right) \cdot 2^{h+1-j(t+1)} \\
&= 2^{h+1} + \sum_{j=1}^{\left\lfloor \frac{h}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-j\cdot t}{j} \cdot 2^{h+1-j(t+1)} \\
&\qquad + \sum_{j=1}^{\left\lfloor \frac{h+1}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-j\cdot t}{j-1} \cdot 2^{h+1-j(t+1)} \quad .
\end{aligned}$$

In case of $\left\lfloor \frac{h+1}{t+1} \right\rfloor = \frac{h+1}{t+1} = \left\lfloor \frac{h}{t+1} \right\rfloor + 1$,

$$\begin{aligned} f(h+1,t) &= 2^{h+1} + \sum_{j=1}^{\left\lfloor \frac{h}{t+1} \right\rfloor} (-1)^j \cdot \binom{h+1-j\cdot t}{j} \cdot 2^{h+1-j(t+1)} \\ &\quad + (-1)^{\frac{h+1}{t+1}} \cdot \binom{h+1-\frac{h+1}{t+1}\cdot t}{\frac{h+1}{t+1}} \cdot 2^{h+1-\frac{h+1}{t+1}(t+1)} \\ &\overset{①②}{=} 2^{h+1} + \sum_{j=1}^{\left\lfloor \frac{h}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-j\cdot t}{j} \cdot 2^{h+1-j(t+1)} \\ &\quad + \sum_{j=1}^{\left\lfloor \frac{h}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-j\cdot t}{j-1} \cdot 2^{h+1-j(t+1)} \\ &\quad + (-1)^{\frac{h+1}{t+1}} \cdot \binom{h-\frac{h+1}{t+1}\cdot t}{\frac{h+1}{t+1}-1} \cdot 2^{h+1-\frac{h+1}{t+1}(t+1)} \\ &= 2^{h+1} + \sum_{j=1}^{\left\lfloor \frac{h}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-j\cdot t}{j} \cdot 2^{h+1-j(t+1)} \\ &\quad + \sum_{j=1}^{\left\lfloor \frac{h+1}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-j\cdot t}{j-1} \cdot 2^{h+1-j(t+1)} \quad . \end{aligned}$$

Thus, we continue for the general case.

$$\begin{aligned} f(h+1,t) &\overset{③}{=} 2^{h+1} + \sum_{j=1}^{\left\lfloor \frac{h}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-j\cdot t}{j} \cdot 2^{h+1-j(t+1)} \\ &\quad + \sum_{j=1}^{\left\lfloor \frac{h-t}{t+1} \right\rfloor+1} (-1)^j \cdot \binom{h-j\cdot t}{j-1} \cdot 2^{h+1-j(t+1)} \\ &= 2 \cdot \left( 2^h + \sum_{j=1}^{\left\lfloor \frac{h}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-j\cdot t}{j} \cdot 2^{h-j(t+1)} \right) \\ &\quad - \left( 2^{h-t} + \sum_{j=1}^{\left\lfloor \frac{h-t}{t+1} \right\rfloor} (-1)^j \cdot \binom{h-t-j\cdot t}{j} \cdot 2^{h-t-j(t+1)} \right) \\ &= 2 \cdot f(h,t) - f(h-t,t) \end{aligned}$$

□

Table 4.3 lists the number $f(h,t)$ of different realizable paths for $h \leq 9$ humping steps and for $t \leq 9$ sorting tracks. Note that for all input sequences with length $n \leq f(\bar{h},\bar{t})$ there is an optimal (feasible) schedule with $h \leq \bar{h}$ humping steps and $t \leq \bar{t}$ sorting tracks for any $t$-**st,ub**$|h$-**hsh,co,sp**$|\cdot,\cdot$ version.

| number $h$ of humping steps | number $t$ of sorting tracks | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 3 | 4 | 7 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 4 | 5 | 12 | 15 | 16 | 16 | 16 | 16 | 16 | 16 |
| 5 | 6 | 20 | 28 | 31 | 32 | 32 | 32 | 32 | 32 |
| 6 | 7 | 33 | 52 | 60 | 63 | 64 | 64 | 64 | 64 |
| 7 | 8 | 54 | 96 | 116 | 124 | 127 | 128 | 128 | 128 |
| 8 | 9 | 88 | 177 | 224 | 244 | 252 | 255 | 256 | 256 |
| 9 | 10 | 143 | 326 | 432 | 480 | 500 | 508 | 511 | 512 |

**Table 4.3.** Number $f(h,t)$ of different realizable paths for $h \leq 9$ humping steps and for $t \leq 9$ sorting tracks

### 4.4.1 Unbounded Case

Observations 4.1 and 4.2 together with Corollary 4.3 imply the validity of Algorithm 7 and of Algorithm 8 for solving the two versions $\underline{t}$-**st,ub**|$h$-**hsh,co,sp**|{**or,fr**},$g$-**bl**, as well as the two versions $t$-**st,ub**|$\underline{h}$-**hsh,co,sp**|{**or,fr**},$g$-**bl**, respectively.

**Algorithm 7:** Greedy for versions
$\underline{t}$-**st,ub**|$h$-**hsh,co,sp**|**or,**$g$-**bl**,
$\underline{t}$-**st,ub**|$h$-**hsh,co,sp**|**fr,**$g$-**bl**

**Input** : An integer sequence $S_{n,g}$
**Output**: An optimal schedule

**Step 1:** Determine a minimum partition of the input sequence $S_{n,g}$ into subsequences $S_1, \ldots, S_{p^*}$ such that $S_1 \oplus S_2 \oplus \cdots \oplus S_{p^*}$ has the structure **ordered** $g$-**blocks** (**free** $g$-**blocks**).

**Step 2:** Compute the minimum number $t^*$ ($\leq h$) of sorting tracks with at least $p^*$ different realizable paths, i. e., with $f(h, t^*) \geq p^*$.
If no such $t^*$ exists, then the problem is infeasible, STOP.

**Step 3:** For all $p = 1, \ldots, p^*$ assign a suitable path $p$ regarding the cyclic track execution order $\mathcal{E}^c = (1, \ldots, t^*)$ to all units in subsequence $S_p$.

**Theorem 4.13**
*Version* $\underline{t}$-***st,ub***|$h$-***hsh,co,sp***|***or,***$g$-***bl*** *is optimally solvable in* $\mathcal{O}(n \log h)$ *time.*

*Proof.* Algorithm 7 computes an optimal solution for version $\underline{t}$-**st,ub**|$h$-**hsh,co,sp**|**or,**$g$-**bl** in $\mathcal{O}(n)$ time: Step 1 can be solved in $\mathcal{O}(n)$, see Subsection 4.2.1 and Algorithm 3. In Step 2, the minimum number $t^*$ of tracks can be computed recursively in $\mathcal{O}(\log h \cdot h)$, see Corollary 4.3. In Step 3, each path $p$ is implicitly given by the reversed binary code of the number $p - 1$. □

Transforming the implicitly given solution to an explicit, easily readable schedule in the shape desired by our practical partner takes $\mathcal{O}(n \cdot h^*)$ time, see Algorithm 9.

**Algorithm 8:** Greedy for versions
$t$-**st,ub**|$\underline{h}$-**hsh,co,sp**|**or,**$g$-**bl**,
$t$-**st,ub**|$\underline{h}$-**hsh,co,sp**|**fr,**$g$-**bl**

**Input** : An integer sequence $S_{n,g}$
**Output**: An optimal schedule

**Step 1:** Determine a minimum partition of the input sequence $S_{n,g}$ into subsequences $S_1, \ldots, S_{p^*}$ such that $S_1 \oplus S_2 \oplus \cdots \oplus S_{p^*}$ has the structure **ordered** $g$-**blocks** (**free** $g$-**blocks**).

**Step 2:** Compute the minimum number $h^*$ of humping steps with at least $p^*$ different realizable paths, i. e., with $f(h^*, t) \geq p^*$.

**Step 3:** For all $p = 1, \ldots, p^*$ assign a suitable path $p$ regarding the cyclic track execution order $\mathcal{E}^c = (1, \ldots, t, 1, \ldots)$ to all units in subsequence $S_p$.

**Theorem 4.14**
*Version* $t$-***st,ub***|$\underline{h}$-***hsh,co,sp***|***or,***$g$-***bl*** *is optimally solvable in* $\mathcal{O}(n \cdot h^*)$ *time.*

*Proof.* Algorithm 8 computes an optimal schedule for version $t$-**st,ub**|$\underline{h}$-**hsh,co,sp**|**or,**$g$-**bl** in $\mathcal{O}(n \cdot h^*)$ time: Step 1 can be solved in $\mathcal{O}(n)$, see Subsection 4.2.1 and Algorithm 3. In Step 2, the minimum number $h^*$ of humping steps can be computed in $\mathcal{O}(n)$, see Corollary 4.3. In Step 3, the first different paths $1, \ldots, p^*$ with respect to the cyclic track execution order $\mathcal{E}^c$ can be generated with Algorithm 9 in $\mathcal{O}(p^* \cdot h^*)$ time. It holds $h^* < p^* \leq n$. □

In other words, the two versions $\underline{t}$-**st,ub**|$h$-**hsh,co,sp**|**or,**$g$-**bl** and $t$-**st,ub**|$\underline{h}$-**hsh,co,sp**|**or,**$g$-**bl** are optimally solvable in linear time in the size of the schedule to compute.

**Theorem 4.15**
*The versions* $\underline{t}$-***st,ub***|$h$-***hsh,co,sp***|***fr,***$g$-***bl***, $t$-***st,ub***|$\underline{h}$-***hsh,co,sp***|***fr,***$g$-***bl*** *are* $\mathcal{NP}$*-hard.*

*Proof.* Let us suppose, that it is decidable in polynomial time whether there exists a feasible solution for the particular version

1-**st,ub**|$(h-1)$-**hsh,co,sp**|**fr,**$g$-**bl**, then, due to previous results and due to the fact that $f(h-1,1) = h$, see proof of Theorem 4.16, we can decide in polynomial time whether the input sequence $S$ can be partitioned into subsequences $S_1, \ldots, S_h$ such that $S_1 \oplus S_2 \oplus \cdots \oplus S_h$ has the structure **free** $g$-**blocks**. However, the latter decision problem is known to be $\mathcal{NP}$-complete, see DAHLHAUS ET AL. (2000a). Thus, 1-**st,ub**|$h$-**hsh,co,sp**|**fr,**$g$-**bl** is $\mathcal{NP}$-complete. □

**Algorithm 9:** Generation of paths for cyclic track execution

**Input** : A cyclic track execution order $\mathcal{E}^c = (e_1, \ldots, e_h)$ of $t$ tracks
**Output**: "*First*" $p^*$ different paths w. r. t. $\mathcal{E}^c$ in $\mathcal{O}(p^* \cdot h)$ time

```
1  Path [ 1 ][ 1 ] ← ··· ← Path [ 1 ][ h* ] ← 0, p ← sub ← 2
2  while p ≤ p* do
3      lastOne ← duplicate ← 0, tmp ← 1
4      for h ← 1 to h* do
5          if h > 1 then tmp ← 2 · tmp
6          if ⌊(sub−1)/tmp⌋ mod 2 = 1 then
7              if h − lastOne ≥ t + 1 then
8                  duplicate ← 1
9                  goto line 12
10             lastOne ← h
11         Path [ p ][ h ] = (⌊(sub−1)/tmp⌋ mod 2) · e_h
12     if duplicate = 0 then p ← p + 1
13     sub ← sub + 1
```

Note that it is possible to determine a feasible solution of the version $t$-**st,ub**|$\underline{h}$-**hsh,co,sp**|**fr,**$g$-**bl** by executing Algorithm 8 with a relaxed Step 1. Instead of computing a minimum partition in Step 1, any partition of the input sequence $S_{n,g}$ into subsequences $S_1, \ldots, S_{\bar{p}}$ such that $S_1 \oplus S_2 \oplus \cdots \oplus S_{\bar{p}}$ has the structure **free** $g$-**blocks** leads to a feasible schedule.

**Theorem 4.16**
*The version* 1-***st,ub***|$\underline{h}$-***hsh,co,sp***|***fr,***$g$-***bl*** *is 2-approximable and the version* $t$-***st,ub***|$\underline{h}$-***hsh,co,sp***|***fr,***$g$-***bl*** *is +2-approximable for* $t \geq 2$*, both in* $\mathcal{O}(n \log n)$ *time.*

*Proof.* We start the proof by inductively showing two results on the number $f(h,t)$ of different realizable paths w. r. t. cyclic track execution.

Assuming $f(\tilde{h},1) = \tilde{h}+1$ for $1 \leq \tilde{h} \leq h$ leads – with our recursive formula – to $f(h+1,1) = 2 \cdot f(h,1) - f(h-1,1) = 2 \cdot (h+1) - h = h+2$. Since $f(1,1) = 1$ and $f(2,1) = 2$, we get $f(h,1) = h+1$ ① for any $h$.

Secondly, let us consider the case $t \geq 2$. If we for $h \geq 2$ assume $f(h,t) > f(h-1,t) + f(h-2,t)$, then, it follows

$$\begin{aligned} f(h+1,t) &= 2 \cdot f(h,t) - f(h-t,t) \\ &\underset{t \geq 2}{\geq} 2 \cdot f(h,t) - f(h-2,t) \\ &> f(h,t) + f(h-1,t) \end{aligned} \quad .$$

Because $f(h,t) > f(h-1,t) + f(h-2,t)$ for $2 \leq h \leq t$, we get $f(h,t) > f(h-1,t) + f(h-2,t)$ for any $h \geq 2$ and $t \geq 2$. As a consequence, $f(h,t) > 2 \cdot f(h-2,t)$ ② for $h \geq 2$ and $t \geq 2$.

The solution of version $\underline{t}$**-qu,ub|nsh,se,**$0$**-sp|fr,**$g$**-bl** – which is computable in $\mathcal{O}(n \log n)$ time, see Subsection 4.3.1 – corresponds to a partition of the input sequence $S_{n,g}$ into subsequences $S_1, \ldots, S_{\bar{p}}$ such that $S_1 \oplus S_2 \oplus \cdots \oplus S_{\bar{p}}$ has the structure **free** $g$**-blocks**.

As above-mentioned, this solution can be transformed – by executing Steps 2 and 3 of Algorithm 8 in $\mathcal{O}(n \cdot \bar{h})$ time – to a solution of $t$**-st,ub|**$\underline{h}$**-hsh,co,sp|fr,**$g$**-bl** with $\bar{h}$ humping steps, that is, $f(\bar{h}-1,t) < \bar{p} \leq f(\bar{h},t)$ ③. It follows a comparison of this solution with the optimal one requiring $h^*$ humping steps and $p^*$ different paths of the units, that is, $f(h^*-1,t) < p^* \leq f(h^*,t)$ ④. Let us assume that $\bar{h} > 2 \cdot h^*$ ⑤ in case $t = 1$ and that $\bar{h} > h^* + 2$ ⑥ in case $t \geq 2$. The proof is finished by showing that in both cases the respective assumption leads to a contradiction with $\bar{p} \leq 2 \cdot p^* - 1$, see Theorem 4.1.

Case $t = 1$:

$$\bar{p} \overset{①③}{=} \bar{h}+1 \overset{⑤}{\geq} 2 \cdot h^* + 2 \overset{①④}{=} 2 \cdot p^*$$

Case $t \geq 2$:

$$\bar{p} \overset{③}{>} f(\bar{h}-1,t) \overset{⑥}{\geq} f(h^*+2,t) \overset{②}{>} 2 \cdot f(h^*,t) \overset{④}{\geq} 2 \cdot p^* \qquad \square$$

### 4.4.2 Bounded Case

The version $t$-**st,**$b$-**bd**|$\underline{h}$-**hsh,co,sp**|**or,**$n$-**bl** is proven to be $\mathcal{NP}$-hard by JACOB ET AL. (2007), a result which extends to the respective $g$-**blocks** versions as well as to the corresponding $t$-**minimizing** versions.

The computational complexity of above versions for forming one outbound train carry over to the respective versions concerning several outbound trains, see Observations 3.4 and 3.12. In particular, version $\underline{t}$-**st,ub**|$h$-**hsh,co,sp**|$o$-**or,**$g$-**bl** is solvable in $\mathcal{O}(n \log h)$ time and version $t$-**st,ub**|$\underline{h}$-**hsh,co,sp**|$o$-**or,**$g$-**bl** can be solved in $\mathcal{O}(n \cdot h^*)$ time. Besides that, the $b$-**bounded** versions $t$-**st,**$b$-**bd**|$\underline{h}$-**hsh,co,sp**|$o$-**or,**$g$-**bl**, $\underline{t}$-**st,**$b$-**bd**|$h$-**hsh,co,sp**|$o$-**or,**$g$-**bl**, as well as our real-world optimization problem $t$-**st,**$b$-**bd**|$\underline{h}$, $\underline{r}$-**hsh,co,sp**|$o$-**or,**$g$-**bl** are $\mathcal{NP}$-hard. JACOB ET AL. (2011) introduce a 2-approximation for $t$-**st,**$b$-**bd**|$\underline{h}$-**hsh,co,sp**|$o$-**or,**$n$-**bl** and HAUSER & MAUE (2010) compare heuristics for improving the schedule obtained by this 2-approximation.

CHAPTER 5

# Coloring Polygon-Circle Graphs

In the previous chapter we have shown that five $t$-**minimizing**, **unbounded**, **no shunting**, $g$-**blocks** versions of SRS are equivalent to the $\mathcal{NP}$-hard MVC of polygon-circle graphs. In other words, one can solve all of these versions by determining a minimum vertex coloring of a particular polygon-circle graph. The corresponding polygon representation is easily constructible for any given instance of our versions, see the proofs of the respective theorems in Subsection 4.3.1. In view of quickly solving the mentioned versions, we deal with MVC of polygon-circle graphs in this chapter. In particular, we introduce a new network flow formulation with side constraints as binary program which exploits the geometrical structure of the polygons. Based on this formulation, a branch-and-bound method is presented. We answer the following two questions. How does this method compete with another branch-and-bound method that is based on a classical assignment formulation regarding computational time? Can the polygon-circle graphs that relate to our real-world instances of the mentioned versions quickly be colored with minimum number of colors?

## 5.1 Approximation

MVC of polygon-circle graphs cannot be approximated within a factor of $2\omega - 1$ where $\omega$ is the clique number, see UNGER (1990) (in German) and UNGER (1992). Besides that, UNGER (1990) – also see UNGER (1988) – claims to have obtained a polynomial time algorithm which produces a minimum vertex coloring of a given circle graph with

no more than $2\omega$ colors, which would prove MVC of circle graphs to be 2-approximable. However, his proof is falsified by a counter-example – see Figure 5.1[1] – given by AGEEV (1996) that consists in a triangle-free circle graph with chromatic number 5. After all, the question of whether or not MVC of circle graphs is 2-approximable remains open. The best known approximation for MVC of circle graphs and of polygon-circle graphs so far has performance guarantee of $\mathcal{H}_{\alpha(G)}$, see Subsection 4.3.2.

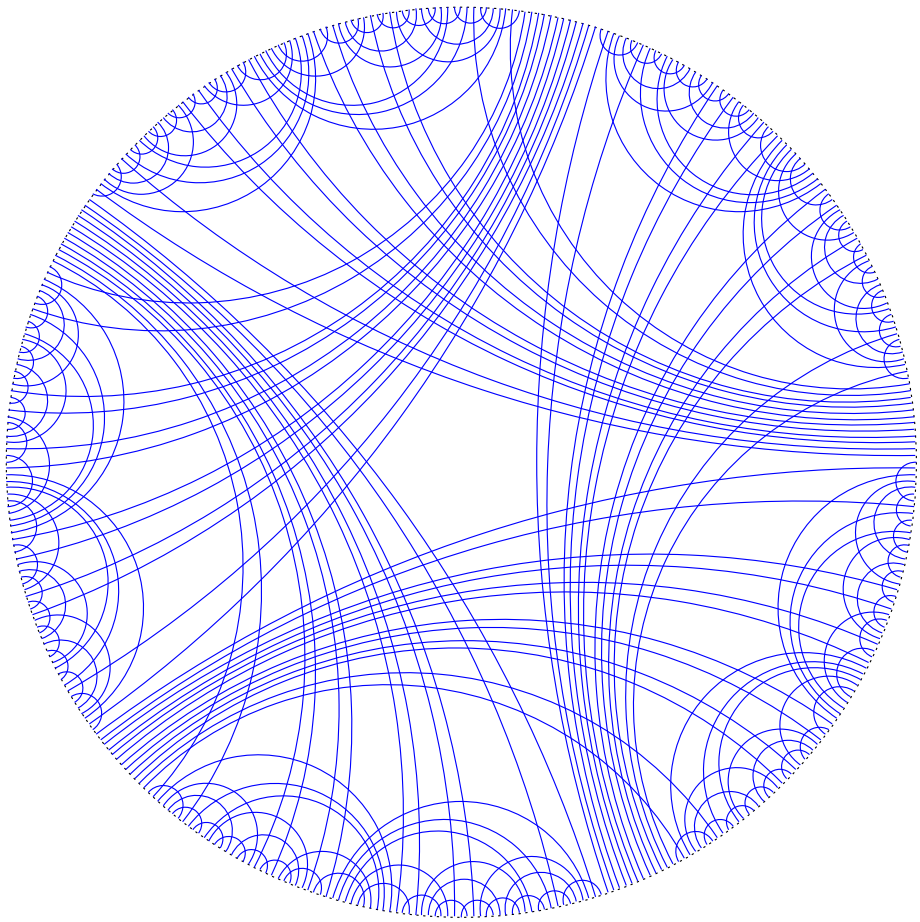

**Figure 5.1.** A triangle-free circle graph with 220 vertices that requires five colors

[1]Thanks to David Eppstein for plotting the graph

## 5.2 Preprocessing

The key idea of a well-known preprocessing for MVC of arbitrary graphs – see Algorithm 10 – is the following: extract an induced subgraph $G[V']$ of the given graph $G = (V, E)$ with $\chi(G[V']) = \chi(G)$, such that we can easily transform a minimum vertex coloring of $G[V']$ into a minimum vertex coloring of the original graph $G$.

**Algorithm 10:** Preprocessing for MVC

**Input** : An arbitrary graph $G = (V, E)$ and a lower bound $\underline{\chi}$ on $\chi(G)$
**Output**: An induced subgraph $G[V']$ of $G$ with above-mentioned property

1 $V' \leftarrow V$
2 **repeat**
3 | Remove each vertex $u \in V'$ from $V'$ if there exists another vertex $v \in V'$ with $N_{G[V']}(u) \subseteq N_{G[V']}(v)$
4 | Remove each vertex $u \in V'$ from $V'$ with $|N_{G[V']}(u)| \leq \underline{\chi} - 1$
5 **until** no more removable vertices are found

It is easily seen that we can augment any feasible coloring of the graph $G[V']$ with $k$ colors produced by Algorithm 10 to a feasible coloring of $G$ with $\max\{\omega, k\}$ colors. This preprocessing is motivated by the following rule of thumb that mostly holds: the smaller the graph, the faster one can compute good or optimal colorings. A lower bound $\underline{\chi}$ that is close to $\chi$ might significantly reduce $G$ to $G[V']$. For most graphs, the clique number $\omega$ is such a good lower bound on $\chi$. For polygon-circle graphs, we compute a maximum clique of size $\omega$ in $\mathcal{O}(n^4)$ time – see Subsection 2.4.1 – and perform the preprocessing with $\underline{\chi} := \omega$ in Algorithm 10.

## 5.3 Heuristics

To provide our exact solution methods with good start solutions we implemented three heuristics for MVC of polygon-circle graphs:

| | |
|---|---|
| FIRST FIT, | see Subsection 2.2.2, |
| MAXIS, | see Algorithm 6 ($b = n$) in Subsection 4.3.2, |
| PACK LONGEST, | see Algorithm 11. |

**Algorithm 11:** PACK LONGEST for MVC of polygon-circle graphs

**Input** : A polygon-circle graph $G^{PC}(\tilde{\mathcal{P}})$ and its polygon representation $\tilde{\mathcal{P}}$ whose polygons are increasingly numbered according to their first corner points

**Output**: A feasible vertex coloring of $G^{PC}(\tilde{\mathcal{P}})$

1 Scale the polygons such that any two consecutive corner points are unit-distant
2 **while** *possible* **do**
3 Pack longest spanned polygon $\mathcal{P}_i \in \tilde{\mathcal{P}}$ in the polygon $\mathcal{P}_j \in \tilde{\mathcal{P}}$ spanning $\mathcal{P}_i$ with smallest index $j$, i. e., the polygons $\mathcal{P}_i$ and $\mathcal{P}_j$ will be assigned the same color
4 Remove $\mathcal{P}_i$ from $\tilde{\mathcal{P}}$
5 Color the remaining interval graph w. r. t. $\tilde{\mathcal{I}} = \{\mathcal{I}^{\mathcal{P}_j} \mid \mathcal{P}_j \in \tilde{\mathcal{P}}\}$ with FIRST FIT and accordingly assign the colors to the polygons

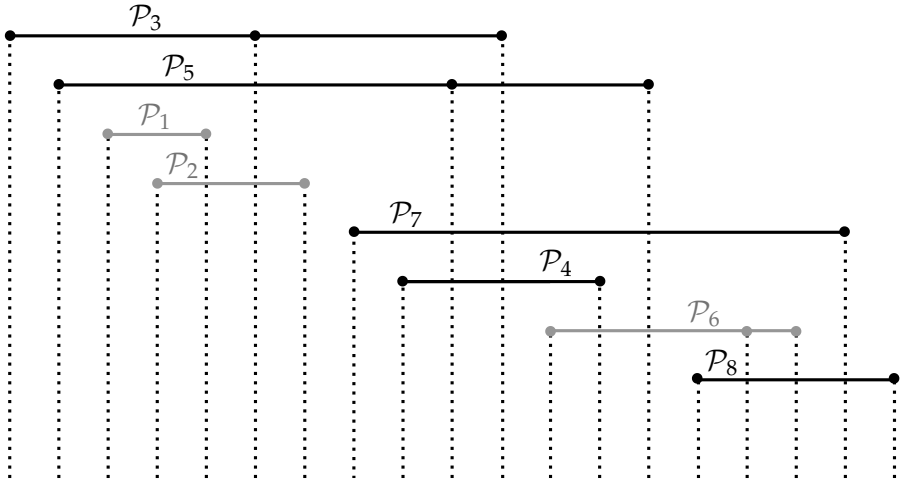


**Figure 5.2.** PACK LONGEST: Polygons with unit-distant corner points

Note, while FIRST FIT and MAXIS only need the polygon-circle graph as input, PACK LONGEST also uses the geometrical structure of

the polygons. Due to the rather informal description of PACK LONGEST in Algorithm 11, we give a little example. Let us consider the polygon representation illustrated in Figure 5.2 after scaling (line 1) and the corresponding polygon-circle graph.

While performing the while-loop (lines 2–4), we iteratively pack $\mathcal{P}_6$ in $\mathcal{P}_7$, $\mathcal{P}_2$ in $\mathcal{P}_5$, and $\mathcal{P}_1$ in $\mathcal{P}_3$. Note, the word "*possible*" in line 2 means that we may only pack a polygon $\mathcal{P}_i$ in its spanning polygon $\mathcal{P}_j$ if all polygons that are already packed in $\mathcal{P}_j$ do not intersect $\mathcal{P}_i$. FIRST FIT colors the intervals $\mathcal{I}^{\mathcal{P}_3}$, $\mathcal{I}^{\mathcal{P}_5}$, $\mathcal{I}^{\mathcal{P}_7}$, $\mathcal{I}^{\mathcal{P}_4}$, and $\mathcal{I}^{\mathcal{P}_8}$ with colors 1, 2, 3, 4, and 1, respectively. After all, PACK LONGEST colors the polygons $\mathcal{P}_3$, $\mathcal{P}_5$, $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_7$, $\mathcal{P}_4$, $\mathcal{P}_6$, and $\mathcal{P}_8$ with the colors 1, 2, 1, 2, 3, 4, 3, 1, respectively. It is easily seen that this coloring is not optimal as $\chi = 3$.

For the preprocessed polygon-circle graphs which relate to our real-world instances of SRS, the quality of the results obtained by above heuristics is compared in Section 6.1.

## 5.4 Exact Solution Methods

In this section we describe three branch-and-bound implementations for solving MVC of polygon-circle graphs that are based on two different BP formulations.

Of course, for solving MVC of polygon-circle graphs we may apply any exact solution approach for MVC that works for arbitrary graphs. Due to its practical relevance, MVC (of arbitrary graphs) is one of the most popular combinatorial optimization problems. As a consequence, there is a broad literature on this problem; a selection of recent publications reads: MEHROTRA & TRICK (1996), CAMPÊLO ET AL. (2004, 2008), MÉNDEZ-DÍAZ & ZABALA (2001, 2006, 2008), COLL ET AL. (2002), CARAMIA & DELL'OLMO (2001, 2004), LUCET ET AL. (2006), PALUBECKIS (2008), HANSEN ET AL. (2009). Most of the proposed exact solution methods are LP based branch-and-cut implementations. They relate to different BP formulations of MVC and make use of corresponding polyhedral results like valid inequalities (cutting planes). It is not the scope of this thesis to provide a thorough comparison and discussion of all exact methods that are presented in the literature. We decided to implement one branch-and-bound method that is based on an existing BP formulation for MVC, see next subsection. The corresponding results are considered as benchmark for our new approach described in Subsection 5.4.2.

### 5.4.1 Assignment Formulation

The following BP is a well-known assignment formulation which probably most intuitively models MVC. By solving this BP one can compute minimum vertex colorings of arbitrary graphs $G = (V, E)$ with $V = \{1, \ldots, n\}$ and $|E| = m$.

$$\min \sum_{k=1}^{\bar{\chi}} y_k \tag{5.1}$$

$$\sum_{\substack{k: \\ (i,k)\in\mathfrak{X}}} x_{i,k} = 1 \qquad \forall i \in V \tag{5.2}$$

$$x_{i,k} + x_{j,k} \leq y_k \qquad \forall k, \{i,j\} \in E : (i,k), (j,k) \in \mathfrak{X} \tag{5.3}$$

$$y_k - y_{k+1} \geq 0 \qquad k = 1, \ldots, \bar{\chi} - 1 \tag{5.4}$$

$$x_{i,k},\ y_k \in \{0,1\} \quad \forall (i,k) \in \mathfrak{X} \tag{5.5}$$

The model contains binary decision variables $x_{i,k}$ and $y_k$. In particular, $x_{i,k} = 1$ if and only if vertex $i$ is colored with color $k$; and $y_k = 1$ if and only if color $k$ is used. By $\bar{\chi}$ we denote an upper bound for the minimum number of colors needed, which is usually determined by heuristics. The set $\mathfrak{X}$ contains all pairs of indices $(i,k)$ for which the variable $x_{i,k}$ exists in the model. Restriction (5.2) ensures that each vertex is colored with exactly one color; (5.3) prevents that adjacent vertices get the same color and also it couples the $x_{i,k}$ and the $y_k$ variables; and (5.4) breaks some symmetry. Of course, above BP is a valid formulation of MVC if $\mathfrak{X} = \{1, \ldots, n\} \times \{1, \ldots, \bar{\chi}\}$. However, dropping variables – without losing the validity of the BP – might significantly reduce the computational time for solving it. Assume that we have determined a large clique $C = \{c_1, \ldots, c_l\}$ of graph $G$. We arbitrarily choose $\bar{\chi} - l - 1$ vertices $v_1, \ldots, v_{\bar{\chi}-l-1}$ that are not contained in $C$ and we define $\bar{V} := V \backslash \{C \cup \{v_1\} \cup \cdots \cup \{v_{\bar{\chi}-l-1}\}\}$. Then, setting $\mathfrak{X}$ to

$$\left( \bigcup_{i=1,\ldots,l} \{(c_i, i)\} \cup \bigcup_{i=1,\ldots,\bar{\chi}-l-1} \bigcup_{j=1,\ldots,l+i} \{(v_i, j)\} \cup \bar{V} \right) \times \{1, \ldots, \bar{\chi}\} \tag{5.6}$$

also leads to a valid formulation of MVC. In other words, the larger the clique $C$ and the lower the upper bound $\bar{\chi}$ the less decision variables

are necessary. For polygon-circle graphs, $\mathfrak{X}$ is easily determined by the largest clique $C$; a maximum clique is computable in $\mathcal{O}(n^4)$ time, see Subsection 2.4.1.

In view of exact solution approaches, we are interested in strengthening the LP relaxation of above BP formulation. For example, this is achieved by replacing (5.3) by the following *clique constraints*

$$\sum_{\substack{i: \\ i \in C \wedge (i,k) \in \mathfrak{X}}} x_{i,k} \leq y_k \qquad \forall k, C \in \mathfrak{C}, \tag{5.7}$$

where $\mathfrak{C} = \{C_{i,j} \mid i, j \in V \text{ with } \{i,j\} \in E\}$ and $C_{i,j}$ is the maximum clique among all cliques that contain the nodes $i$ and $j$. For any two adjacent vertices $i$ and $j$, we obtain $C_{i,j}$ by computing a maximum clique in $G[N_G(i) \cap N_G(j)]$. For polygon-circle graphs, the worst-case complexity for generating $\mathfrak{C}$ is $\mathcal{O}(mn^4)$, see Subsection 2.4.1. For large instances we might fail to compute $\mathfrak{C}$ entirely in adequate time. Thus, it is reasonable to impose a time and space limit for generating $\mathfrak{C}$. For those pairs of adjacent vertices $i$ and $j$, where we could not determine $C_{i,j}$ within the given amount of time, we add the respective restriction of (5.3) to the model; for all other pairs $i$ and $j$ we add the constraint of (5.7) with respect to $C_{i,j} \in \mathfrak{C}$. We denote the resulting BP formulation as ASSIGNMENT BP.

Let us now describe the relevant ingredients of the implemented LP based branch-and-bound procedure that is based on the ASSIGNMENT BP; for a general description of branch-and-bound, see Section 2.2.2.

### LP based branch-and-bound (ASSIGNMENT)

**Preprocessing** We compute an MWC of the given polygon-circle graph which results in the lower bound $\omega$ on $\chi$, and we preprocess the graph as described in Section 5.2. Among the three feasible colorings computed by the proposed heuristics, we choose the best coloring, which provides an initial upper bound on $\chi$. Besides that, the preprocessed graph is decomposed into its connected components. For each of these components, we – in some sense – run a separate branch-and-bound procedure to compute a sufficiently good coloring.

**Bounding** A lower bound on the optimal value $z^*_{\mathfrak{s}}$ of the currently processed subproblem $\mathfrak{P}_{\mathfrak{s}}$ is obtained by solving the corresponding LP relaxation; in particular, the lower bound equals $\lceil z^{\text{LP}}_{\mathfrak{s}} \rceil$ where $z^{\text{LP}}_{\mathfrak{s}}$ is the value of the LP solution. An upper bound on $z^*_{\mathfrak{s}}$ may possibly be determined either by an integral LP solution or by the ITERATIVE ROUNDING scheme. This procedure – illustrated in Algorithm 12 – is also known as FRACTIONALITY DIVING. Given an LP solution, we define the set of indices $\hat{\mathfrak{X}} := \{(i,k) \in \mathfrak{X} \mid 0 < \hat{x}_{i,k} < 1\}$ of the variables $x_{i,k}$ with fractional values $\hat{x}_{i,k}$. The greatest fractional value $\max_{(i,k)\in\hat{\mathfrak{X}}}\{\hat{x}_{i,k}\}$ is denoted by $f_{\max}$.

**Algorithm 12:** ITERATIVE ROUNDING

**Input** : LP relaxation of the current subproblem, parameter $0 \leq \delta < 1$

**Output**: If available, a feasible coloring

```
1 repeat
2     Solve the LP with current fixings of variables
3     if LP is infeasible then return "no feasible coloring found"
4     if LP solution is integral then return feasible coloring else
      fix all variables x_{i,k} with (i,k) ∈ X̂ and x̂_{i,k} ≥ f_max − δ to 1
```

**Branching Policy** Note, each node in the search tree is characterized by the fixed $x_{i,k}$ variables, and it can be interpreted as a particular partial coloring of the graph. We say a vertex $i$ has color $k$ *available* if the variable $x_{i,k}$ with $(i,k) \in \mathfrak{X}$ is not yet fixed (to 1 or 0). An *uncolored* vertex has at least two colors available. Among all uncolored vertices, we select one vertex according to the following CELIM rule introduced by SEWELL (1993). For all uncolored vertices $i$, we sum up – over all colors $k$ available to $i$ – the number of $i$'s uncolored neighbors that still have color $k$ available. The vertex $i^*$ with the largest sum is then chosen to be the *branching vertex* for the current subproblem $\mathfrak{P}_{\mathfrak{s}}$ with respect to the (solution) subset $X_{\mathfrak{s}}$. That is, we subdivide $X_{\mathfrak{s}}$ as follows: we create a subproblem of $\mathfrak{P}_{\mathfrak{s}}$ for each color $k$ available to $i^*$ in which we additionally fix $x_{i^*,k}$ to 1. Of course, this may lead to further fixings of other *free* – that is, not yet fixed – variables.

**Node Selection Policy** In our implementation, we maintain the list of unpruned nodes (subproblems) in form of the following queue. At

each subdivision of a subproblem $\mathfrak{P}_s$, we insert the subproblem of $\mathfrak{P}_s$ created for the smallest color available to the branching vertex $i^*$ at the head of the queue; and the subproblems of $\mathfrak{P}_s$ generated for the remaining colors available to $i^*$ are inserted – in order of increasing colors – at the tail of the queue. We process the unpruned subproblems in the order of the queue items from head to tail. The resulting node selection policy is a mixture of the well-known depth first search and breadth first search.

### 5.4.2 Network Flow Formulation

In this section we present a new BP formulation for MVC of polygon-circle graphs $G^{PC}(\tilde{\mathcal{P}}) = (V, E)$ with $V = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and $|E| = m$ which exploits the geometrical structure of the polygons. Remember, in $\tilde{\mathcal{P}}$ the polygons are increasingly numbered according to the order of their first corner points, see Convention 2.1. The set $\{1, \ldots, n\}$ is denoted by $\hat{V}$. For all pairs of $i, j \in \hat{V}$, we define

$$\alpha(i,j) := \begin{cases} 1, & \text{if } \mathcal{P}_i \text{ is spanned by } \mathcal{P}_j \\ 2, & \text{if } \mathcal{P}_j \text{ is spanned by } \mathcal{P}_i \\ 3, & \text{if } \mathcal{I}^{\mathcal{P}_i} \lessdot \mathcal{I}^{\mathcal{P}_j} \\ 0, & \text{otherwise} \end{cases} . \tag{5.8}$$

Let us consider the following BP.

$$\min \sum_{i \in \hat{V}} y_{q,i} \tag{5.9}$$

$$y_{i,j} - y_{j,i} = 0 \qquad i,j \in \hat{V}\colon \alpha(i,j) = 2 \tag{5.10}$$

$$y_{i,j} + y_{i,k} \leq 1 \qquad i,j,k \in \hat{V}\colon \alpha(i,j) = \alpha(i,k) = 2, \{\mathcal{P}_j, \mathcal{P}_k\} \in E \tag{5.11}$$

$$y_{i,j} + y_{j,k} \leq 1 \qquad i,j,k \in \hat{V}\colon \alpha(i,j) = \alpha(j,k) = 2 \tag{5.12}$$

$$y_{q,i} + \sum_{\substack{j \in \hat{V}\colon \\ \alpha(j,i) \geq 2}} y_{j,i} = 1 \qquad i \in \hat{V} \tag{5.13}$$

$$y_{i,s} + \sum_{\substack{j \in \hat{V}\colon \\ \alpha(i,j) \geq 1}} y_{i,j} - \sum_{\substack{j \in \hat{V}\colon \\ \alpha(j,i) = 1}} y_{j,i} = 1 \qquad i \in \hat{V} \tag{5.14}$$

$$y_{q,i},\ y_{i,s} \in [0,1] \qquad i \in \hat{V} \tag{5.15}$$

$$y_{i,j} \in [0,1] \qquad i,j \in \hat{V}\colon \alpha(i,j) \in \{1,3\} \tag{5.16}$$

$$y_{i,j} \in \{0,1\} \qquad i,j \in \hat{V}\colon \alpha(i,j) = 2 \tag{5.17}$$

For a given polygon-circle graph, the above BP formulates a particular network flow problem in a network denoted by $\mathcal{N}$. This network contains the source $q$, the sink $s$, and a node $i$ for each polygon $\mathcal{P}_i \in V$. There are arcs from $q$ to all of these nodes $i$, and arcs from all of these nodes $i$ to $s$. Furthermore, there is an arc $(i,j)$ from node $i$ to $j$ if and only if $\alpha(i,j) \geq 1$, that is, there is no arc $(i,j)$ if the polygons $\mathcal{P}_i$ and $\mathcal{P}_j$ intersect or if $\mathcal{I}^{\mathcal{P}_j} \lessdot \mathcal{I}^{\mathcal{P}_i}$. The binary variables $y_{i,j}$, $y_{q,i}$, and $y_{i,s}$ can be interpreted as flow values on the arcs $(i,j)$, $(q,i)$, and $(i,s)$, respectively. For any arc, the lower capacity on the flow value is 0, the upper capacity is 1, see (5.15)–(5.17). Each arc leaving the source occasions costs of 1; no other costs are incurred. The above BP seeks for a particular min cost flow in $\mathcal{N}$. The feasibility of the flow is restricted by additional side constraints that extend a classical min cost flow problem. For a better understanding of these side constraints, we visualize each node $i$ in $\mathcal{N}$ such that – in its interior – the IWIP of polygon $\mathcal{P}_i$ is plotted at the correct position, see Figure 5.3 for an example.

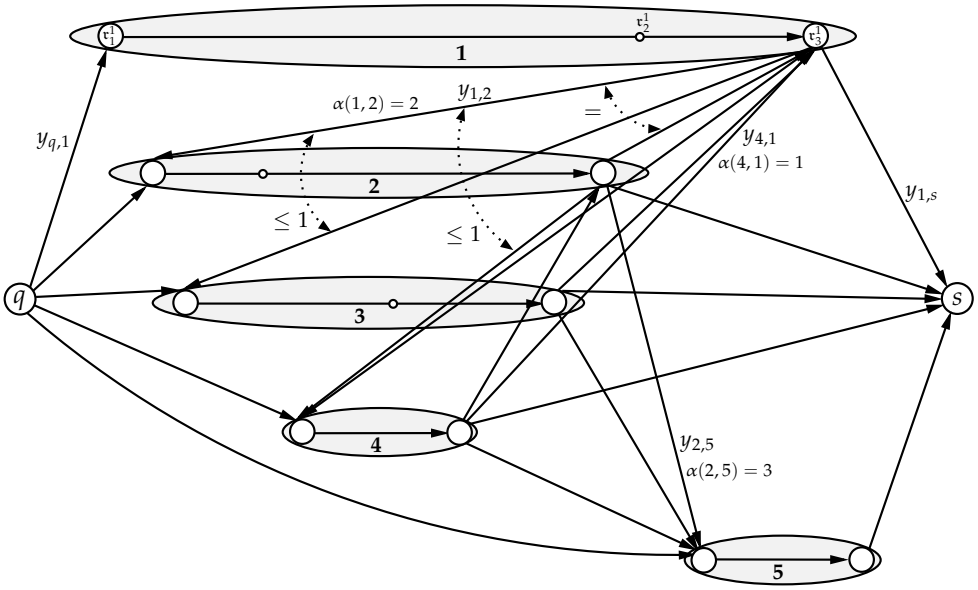


**Figure 5.3.** The network $\mathcal{N}$ for a particular polygon-circle graph

For each node $i$, let the arc $(q,i)$ and the arcs $(j,i)$ with $\alpha(j,i) \geq 2$ enter node $i$ at the first (corner) point $\mathfrak{r}_1^i$; and let the arc $(i,q)$, the arcs $(i,j)$ with $\alpha(i,j) \geq 1$, and the arcs $(j,i)$ with $\alpha(j,i) = 1$ enter or leave node $i$ at the last (corner) point $\mathfrak{r}_{n_i^c}^i$, respectively. Then, the effect of the

restrictions (5.13) and (5.14) – which also formulate the flow conservation at any node $i$ – can be illustrated by fixing the flow value to 1 on an imaginary arc from $\mathfrak{r}_1^i$ to $\mathfrak{r}_{n_i^c}^i$. For any arc $(i,j)$ with $\alpha(i,j) = 2$, (5.17) requires an integral flow value which equals the flow value on arc $(j,i)$, see (5.10). Constraint (5.11) allows a (positive) flow on only one of two arcs $(i,j)$ and $(i,k)$ with $\alpha(i,j) = \alpha(i,k) = 2$ and $\{\mathcal{P}_j, \mathcal{P}_k\} \in E$. Furthermore, restriction (5.12) prohibits a flow on both of two arcs $(i,j)$ and $(j,k)$ with $\alpha(i,j) = \alpha(j,k) = 2$.

Note, if all binary variables are fixed, then the BP described by (5.9)–(5.17) becomes a classical min cost flow problem. Thus, there is an optimal solution of this BP that is integral. In particular, if all binary variables are fixed, then every basic feasible solution of the corresponding LP relaxation is integral. An integral feasible solution of this BP with objective value $z$ corresponds to a *feasible path family* of $z$ $q$-$s$-paths in $\mathcal{N}$. The feasibility of the path family is restricted by the constraints (5.10)–(5.14), and (5.17). In particular, in a feasible path family, each node $i$ is traversed by exactly one path.

**Theorem 5.1**
*Given a polygon representation $\tilde{\mathcal{P}}$, a feasible path family of $z$ $q$-$s$-paths in $\mathcal{N}$ corresponds to a feasible coloring of $G^{PC}(\tilde{\mathcal{P}})$ with $z$ colors.*

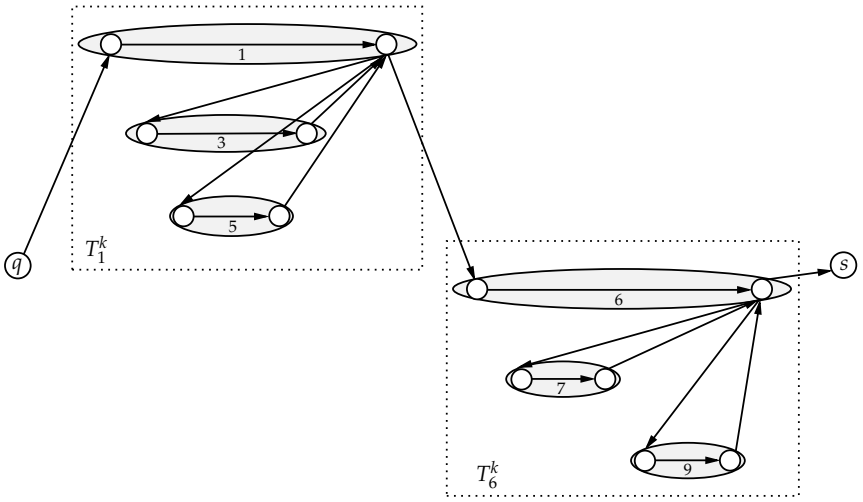


**Figure 5.4.** A path $k$ and the partition of $V^k$ into towers

*Proof.* Let a feasible path family of $z$ $q$-$s$-paths in $\mathcal{N}$ be given. For $k = 1, \ldots, z$, we define $V^k := \{\mathcal{P}_i \in V \mid \text{node } i \text{ is on path } k\}$. Due to the

feasibility of the path family, $\{V^1, \ldots, V^z\}$ is a partition of $V$. It suffices to verify that each $V^k$ is an independent set in $G^{PC}(\tilde{\mathcal{P}})$.

For $i \in \hat{V}$ with $\mathcal{P}_i \in V^k$, set $T_i^k := \{i\} \cup \{\, j \in \hat{V} \mid \mathcal{P}_j \in V^k,\, \alpha(i,j) = 2 \,\}$ is called *tower* of path $k$ if there does not exist a $\mathcal{P}_l \in V^k$ with $\alpha(l,i) = 2$. Let us assume that the set of nodes on path $k$ decomposes into $t$ towers, see Figure 5.4 for an example. If two nodes $i$ and $j$ with $\mathcal{P}_i, \mathcal{P}_j \in V^k$ do not both belong to the same tower, then $\mathcal{P}_i$ and $\mathcal{P}_j$ are obviously not adjacent in $G^{PC}(\tilde{\mathcal{P}})$. Hence, it remains to show the independence within the towers. Let us consider an arbitrary tower $T_l^k$ of path $k$. If $i$ with $i \neq l$ is a node contained in this tower, we know that path $k$ traverses both the arcs $(i,l)$ and $(l,i)$, due to the constraints (5.10) and (5.12). The pure existence of these arcs indicates that $\mathcal{P}_i$ and $\mathcal{P}_l$ are not adjacent in $G^{PC}(\tilde{\mathcal{P}})$. Besides that, if $i$ and $j$ with $i \neq j \neq l$ are two nodes that belong to tower $T_l^k$, we know that path $k$ traverses both the arcs $(l,i)$ and $(l,j)$. However, this is only possible since $\{\mathcal{P}_i, \mathcal{P}_j\} \notin E$, see (5.11). As a consequence, $V^k$ are independent sets in $G^{PC}(\tilde{\mathcal{P}})$. □

**Theorem 5.2**
*If there is a feasible coloring of the polygon-circle graph $G^{PC}(\tilde{\mathcal{P}})$ with $z$ colors, then it exists a feasible path family of $z$ $q$-$s$-paths in $\mathcal{N}$.*

*Proof.* Let a feasible coloring of $G^{PC}(\tilde{\mathcal{P}})$ with $z$ colors be given. Such a coloring corresponds to a partition of $V$ into $z$ independent sets $V^1, \ldots, V^z$. For each $V^k$, we construct a feasible path in $\mathcal{N}$ that passes through all nodes in $\hat{V}^k := \{i \in \hat{V} \mid \mathcal{P}_i \in V^k\}$. Assume that $\hat{V}^k$ decomposes into $\mathfrak{t}$ towers $T_{i_1}^k, \ldots, T_{i_\mathfrak{t}}^k$ with $i_1 < i_2 < \cdots < i_\mathfrak{t}$, see the proof of Theorem 5.1 for the definition of tower. Obviously, the path $(q,i_1)$,$(i_1,i_2)$, $\ldots$ ,$(i_{\mathfrak{t}-1},i_\mathfrak{t})$,$(i_\mathfrak{t},s)$ is feasible. Within all towers $T_i^k$, we extend this path by additionally traversing all arcs $(i,j)$ and $(j,i)$ for all $j \in T_i^k$ with $i \neq j$. This path passes through all nodes contained in $\hat{V}^k$, but no other nodes. Since $V^k$ is an independent set, this path satisfies all constraints – in particular, (5.10), (5.11), and (5.12) – that ensure feasibility. The composition of the paths for $V^1, \ldots, V^z$ results in the desired path family. □

The size of the BP described by (5.9)–(5.17) can be reduced: of course, we can substitute the variables $y_{j,i}$ with $\alpha(j,i) = 1$ by the variables $y_{i,j}$. In order to strengthen the LP relaxation of this BP formulation, we replace (5.11) and (5.12) by the *clique constraints* (5.19). Besides that, we provide the trivial lower bound on the objective value,

that is, on the number of paths, which is given by the size $\omega$ of the maximum clique, see (5.22). The resulting model reads as follows.

$$\min \sum_{i \in \hat{V}} y_{q,i} \tag{5.18}$$

$$\sum_{\substack{j \in \hat{V}: \\ \alpha(j,i)=2}} y_{j,i} + \sum_{j \in C} y_{i,j} \leq 1 \qquad C \in \mathfrak{C} : C = C^i_l \text{ or } C = C^i_{l,k}, \tag{5.19}$$

$$y_{q,i} + \sum_{\substack{j \in \hat{V}: \\ \alpha(j,i) \geq 2}} y_{j,i} = 1 \qquad i \in \hat{V} \tag{5.20}$$

$$y_{i,s} + \sum_{\substack{j \in \hat{V}: \\ \alpha(j,i)=2}} y_{j,i} + \sum_{\substack{j \in \hat{V}: \\ \alpha(i,j)=3}} y_{i,j} = 1 \qquad i \in \hat{V} \tag{5.21}$$

$$\sum_{i \in \hat{V}} y_{q,i} \geq \omega \tag{5.22}$$

$$y_{q,i},\ y_{i,s} \in [0,1] \qquad i \in \hat{V} \tag{5.23}$$

$$y_{i,j} \in [0,1] \qquad i,j \in \hat{V} \colon \alpha(i,j) = 3 \tag{5.24}$$

$$y_{i,j} \in \{0,1\} \qquad i,j \in \hat{V} \colon \alpha(i,j) = 2 \tag{5.25}$$

$\mathfrak{C}$ is the union of $\{C^i_{l,k} \,|\, i,l,k \in \hat{V},\ \alpha(i,l) = \alpha(i,k) = 2,\ \{\mathcal{P}_l, \mathcal{P}_k\} \in E\}$ and $\{C^i_l \,|\, i,l \in \hat{V},\ \alpha(i,l) = 2\}$. $C^i_{l,k}$ is a maximum clique in $G^{PC}[V^i_{l,k}]$, and $C^i_l$ is a maximum clique in $G^{PC}[V^i_l]$, where

$$\begin{aligned} V^i_{l,k} &:= \{\mathcal{P}_l, \mathcal{P}_k\} \cup \{\mathcal{P}_{k_2} \,|\, k_2 \neq l \neq k,\ \alpha(i,k_2) = 2, \\ &\qquad\qquad \{\mathcal{P}_{k_2}, \mathcal{P}_l\}, \{\mathcal{P}_{k_2}, \mathcal{P}_k\} \in E\}, \\ V^i_l &:= \{\mathcal{P}_l\} \cup \{\mathcal{P}_k \,|\, k \neq l,\ \alpha(i,k) = 2,\ \{\mathcal{P}_k, \mathcal{P}_l\} \in E\}. \end{aligned}$$

The worst-case complexity for generating $\mathfrak{C}$ is $\mathcal{O}(n^6)$, see Subsection 2.4.1. Again, we impose a time and space limit for generating $\mathfrak{C}$. In the case that we failed to compute $\mathfrak{C}$ entirely in the given amount of time, our model contains an adequate mixture of the constraints (5.11), (5.12), and (5.19). We denote the resulting formulation as NETWORK BP.

Let us summarize the results that we presented in this subsection so far. There is an optimal solution of the NETWORK BP that is integral. Such an integral solution with objective value $z$ corresponds to a feasible path family of $z$ $q$-$s$-paths in $\mathcal{N}$, and vice versa. A feasible path family of $z$ $q$-$s$-paths in $\mathcal{N}$ is transformable into a feasible coloring of the given polygon-circle graph with $z$ colors, and vice versa. As a consequence, the NETWORK BP is a valid formulation for MVC of polygon-circle graphs.

Based on the NETWORK BP we implemented two branch-and-bound procedures: an LP based method and a Lagrangian approach. For both branch-and-bound procedures, a description of essential subroutines follows. We start with the LP based branch-and-bound, followed by the Lagrangian one.

### LP based branch-and-bound (NETWORK)

**Preprocessing** We perform the same preprocessing as for the LP based ASSIGNMENT branch-and-bound, see page 89.

**Bounding** Again, we implemented the following LP based bounding procedure: a lower bound on the optimal value $z^*_{\mathfrak{s}}$ of the currently processed subproblem $\mathfrak{P}_{\mathfrak{s}}$ is given by $\lceil z^{\text{LP}}_{\mathfrak{s}} \rceil$ where $z^{\text{LP}}_{\mathfrak{s}}$ is the value of the LP solution; an upper bound on $z^*_{\mathfrak{s}}$ may possibly be determined either by an integral LP solution or by the ITERATIVE ROUNDING scheme, see Algorithm 12.

Additionally, we compute feasible colorings and corresponding upper bounds by applying PACK LONGEST. Here, PACK LONGEST starts with a fixed "*pre-packing*" which is given by the binary variables that are fixed to 1 in the current subproblem. Remember, PACK LONGEST attempts to pack spanned polygons in decreasing order of their size, see Algorithm 11. Actually, we determine several feasible colorings by iteratively executing an adapted Algorithm 11. The first iteration corresponds to PACK LONGEST. Afterwards, at each run, we pack spanned polygons in an order that we obtain by slightly altering the order of the previous iteration.

In a leaf node, if the LP solution is not integral, we determine an optimal solution (for this leaf node) which is integral by solving a min cost flow problem in a network that results from the fixings of the binary variables.

**Branching Policy** For each free binary variable $y_{i,j}$ we compute the number $\mathfrak{d}_{i,j}$ of deductions on other free binary variables that result from fixing $y_{i,j}$ to 1. At the same time, we check wether fixing $y_{i,j}$ to 1 leads to infeasibility. If this is the case for any free binary variable, then, we set all free binary variables to 0 and we obtain the optimal solution for the current subproblem by solving a classical min cost flow in the resulting network. Otherwise, we branch on a free binary variable $y_{i,j}$ with greatest $\mathfrak{d}_{i,j}$. This kind of branching is also known as *strong inference branching*.

**Node Selection Policy** As in the ASSIGNMENT BP implementation, we maintain the list of unpruned nodes (subproblems) in form of a queue. At each subdivision of a subproblem $\mathfrak{P}_{\mathfrak{s}}$, we insert the subproblem of $\mathfrak{P}_{\mathfrak{s}}$ that we obtain by fixing the branching variable to 1 at the head of the queue; and the subproblem of $\mathfrak{P}_{\mathfrak{s}}$ in which we fix the branching variable to 0 at the tail of the queue. Again, our node selection policy is to process the unpruned subproblems in the order of the queue items from head to tail.

### Lagrangian branch-and-bound (NETWORK)

**Preprocessing** As before, see page 89.

**Bounding** In this approach, we refrain from the power of LP relaxations and LP solvers. Instead, we obtain bounds by solving a Lagrangian relaxation of the currently processed subproblem $\mathfrak{P}_{\mathfrak{s}}$. Remember, such a subproblem – that is, the NETWORK BP with fixings of some binary variables – amounts to finding a feasible path family of minimum number of paths in the network $\mathcal{N}$. Note that the flow value on some arcs is pre-defined according to the fixings. For any spanned polygon $\mathcal{P}_j$, we define $\mathsf{firstSpanning}(j) := \min\{i \in \hat{V} \mid \alpha(i,j) = 2\}$. It is easily seen that a feasible solution of the Binary Program (5.26)–(5.36) – see next page – describes a path family in $\mathcal{N}$. The paths of such a path family satisfy the constraints (5.10)–(5.12). However, this BP only requires that a node is traversed at least once. Nevertheless, we can obviously shrink these paths such that each node is traversed exactly once; this results in a feasible path family in $\mathcal{N}$ which satisfies (5.10)–(5.14), and (5.17). As a consequence, the optimal value of the Binary Program (5.26)–(5.36) equals the optimal value $z^*_{\mathfrak{s}}$ of subproblem $\mathfrak{P}_{\mathfrak{s}}$.

$$z_{\mathfrak{s}}^{*} = \min \sum_{i \in \hat{V}} y_{q,i} \tag{5.26}$$

$$y_{i,j} + y_{i,k} \leq 1 \qquad \begin{array}{l} i,j,k \in \hat{V}: \\ \alpha(i,j) = \alpha(i,k) = 2, \\ \{\mathcal{P}_j, \mathcal{P}_k\} \in E \end{array} \tag{5.27}$$

$$y_{i,j} + y_{j,k} \leq 1 \qquad \begin{array}{l} i,j,k \in \hat{V}: \\ \alpha(i,j) = \alpha(j,k) = 2, \\ \mathsf{firstSpanning}(j) = i \end{array} \tag{5.28}$$

$$\sum_{\substack{j \in \hat{V}: \\ \alpha(j,i)=2}} y_{j,i} + \sum_{j \in C_l^i} y_{i,j} \leq 1 \qquad \begin{array}{l} i,l \in \hat{V} : \alpha(i,l) = 2, \\ \mathsf{firstSpanning}(l) \neq i \end{array} \tag{5.29}$$

$$y_{q,i} + \sum_{\substack{j \in \hat{V}: \\ \alpha(j,i) \geq 2}} y_{j,i} \geq 1 \qquad i \in \hat{V} \tag{5.30}$$

$$y_{q,i} + \sum_{\substack{j \in \hat{V}: \\ \alpha(j,i)=3}} y_{j,i} \leq 1 \qquad i \in \hat{V} \tag{5.31}$$

$$y_{q,i} + \sum_{\substack{j \in \hat{V}: \\ \alpha(j,i)=3}} y_{j,i} - y_{i,s} - \sum_{\substack{j \in \hat{V}: \\ \alpha(i,j)=3}} y_{i,j} = 0 \qquad i \in \hat{V} \tag{5.32}$$

$$\sum_{i \in \hat{V}} y_{q,i} \geq \omega \tag{5.33}$$

$$y_{q,i} + \sum_{\substack{j \in \hat{V}: \\ \alpha(j,i)=3}} y_{j,i} \geq 1 \qquad \begin{array}{l} i \in \hat{V} : \exists k : \alpha(i,k) = 2, \\ y_{i,k} \text{ is fixed to 1} \end{array} \tag{5.34}$$

$$y_{q,i} + \sum_{\substack{j \in \hat{V}: \\ \alpha(j,i)=3}} y_{j,i} = 0 \qquad \begin{array}{l} i \in \hat{V} : \exists k : \alpha(k,i) = 2, \\ y_{k,i} \text{ is fixed to 1} \end{array} \tag{5.35}$$

$$y_{q,i},\, y_{i,s},\, y_{i,j} \begin{cases} \in & \{0,1\}, \quad \text{variable is free} \\ = & 1, \quad \text{variable is fixed to 1} \\ = & 0, \quad \text{variable is fixed to 0} \end{cases} \tag{5.36}$$

In order to obtain a lower bound on $z_5^*$, we Lagrange relax the constraints (5.29) and (5.30) and solve the resulting Lagrangian dual. For fixed Lagrange multipliers, we can determine an optimal solution of the inner problem by solving two separate subproblems. Note, (5.27) and (5.28) only restrict the variables $y_{i,j}$ with $\alpha(i,j) = 2$. However, these variables do not occur within the constraints (5.31)–(5.35). For the variables $y_{i,j}$ with $\alpha(i,j) = 2$, the subproblem $\mathcal{L}_1$ which reads

$$\min \sum_{\substack{i,j \in \hat{V}: \\ \alpha(i,j)=2}} \tilde{w}_{i,j}\, y_{i,j} \tag{5.37}$$

$$\text{s.t.} \quad (5.27), (5.28), (5.36)$$

computes the optimal values for the inner problem. For the other variables, we obtain the optimal values by solving the subproblem

$$\min \left\{ \sum_{i \in \hat{V}} \tilde{w}_{q,i}\, y_{q,i} + \sum_{\substack{i,j \in \hat{V}: \\ \alpha(i,j)=3}} \tilde{w}_{i,j}\, y_{i,j} \right\} \tag{5.38}$$

$$\text{s.t.} \quad (5.31)–(5.36),$$

which we will refer to as $\mathcal{L}_2$ for short. In these subproblems, $\tilde{w}_{q,i}$ and $\tilde{w}_{i,j}$ denote the weights of the variables in the objective function of the inner problem.

Let us consider the subproblem $\mathcal{L}_1$. It is obvious that $\mathcal{L}_1$ amounts to the problem of finding an MWIS of the graph $\tilde{G} = (\tilde{V}, \tilde{E})$ with $\tilde{V} = \{(i,j) \mid \alpha(i,j) = 2\}$. In this graph, the adjacency of vertices – that is, $\tilde{E}$ – is captured by the constraints (5.27) and (5.28). For all $i \in \hat{V}$ with $\nexists k : \alpha(k,i) = 2$, let us define

$$\begin{aligned} \tilde{V}_i \;:=\; & \{(i,j) \in \tilde{V} \mid \alpha(i,j) = 2\} \cup \\ & \{(j,k) \in \tilde{V} \mid \alpha(i,j) = \alpha(j,k) = 2,\ \mathsf{firstSpanning}(j) = i\}. \end{aligned}$$

For all other $i \in \hat{V}$, we set $\tilde{V}_i := \emptyset$. Note, $\tilde{V}_1, \ldots, \tilde{V}_n$ partitions $\tilde{V}$. It is easily verified that $\{\mathcal{P}_{i,j}(\tilde{\mathfrak{R}}_{i,j}) \mid \alpha(i,j) = 2\}$ – which is determined by Algorithm 13 – is a polygon representation of $\tilde{G}$. In other

words, $\tilde{G}$ is a polygon-circle graph. As a consequence, we can solve $\mathcal{L}_1$ in $\mathcal{O}(|\tilde{V}|^2 + |\tilde{V}||\tilde{E}|)$ time by applying Algorithm 2. Note that Algorithm 13 is a rather understandable scheme for generating the polygon representation of $\tilde{G}$; of course, its principle can be implemented more efficiently by incorporating a linked list for each $\tilde{V}_i$.

**Algorithm 13:** Generation of a polygon representation for $\tilde{G}$

**Input** : Polygon representation $\tilde{\mathcal{P}} = \{\mathcal{P}_1(\mathfrak{R}_1), \ldots, \mathcal{P}_n(\mathfrak{R}_n)\}$ of the original polygon-circle graph, $\epsilon > 0$ sufficiently small, $M > 0$ sufficiently large

**Output**: Polygons $\mathcal{P}_{i,j}(\tilde{\mathfrak{R}}_{i,j})$ for all $(i,j) \in \tilde{V}$

$\mathsf{shift} \leftarrow 0$, $\tilde{\mathfrak{R}}_{i,j} \leftarrow \hat{\mathfrak{R}}_{i,j} \leftarrow \emptyset$ for all $(i,j) \in \tilde{V}$
**for** $i \leftarrow 1$ **to** $n$ **do**
  **if** $\tilde{V}_i \neq \emptyset$ **then**
    **for** $j \leftarrow i+1$ **to** $n$ **do**
      **if** $(i,j) \in \tilde{V}_i$ **then** $\hat{\mathfrak{R}}_{i,j} \leftarrow \mathfrak{R}_j$
      **for** $k \leftarrow j+1$ **to** $n$ **do**
        **if** $(j,k) \in \tilde{V}_i$ **then**
          $\hat{\mathfrak{R}}_{j,k} \leftarrow \left\{ \mathfrak{r}^j_{n^c_j} + \epsilon \cdot \mathfrak{r} \,\middle|\, \mathfrak{r} \in \mathfrak{R}_k \right\}$
          $\hat{\mathfrak{R}}_{i,j} \leftarrow \hat{\mathfrak{R}}_{i,j} \cup \left\{ \mathfrak{r}^j_{n^c_j} + \mathfrak{r}^k_1 + \frac{\epsilon}{M} \right\}$
    **for** all $(j,k) \in \tilde{V}_i$ **do**
      $\tilde{\mathfrak{R}}_{j,k} \leftarrow \{\mathfrak{r} + \mathsf{shift} \mid \mathfrak{r} \in \hat{\mathfrak{R}}_{j,k}\}$
    $\mathsf{shift} \leftarrow \max\{\mathfrak{r} \mid \mathfrak{r} \in \tilde{\mathfrak{R}}_{j,k},\ (j,k) \in \tilde{V}_i\}$

Subproblem $\mathcal{L}_2$ corresponds to a min cost flow problem (with side constraints) in a network $\tilde{\mathcal{N}}$ that contains the source $q$, the sink $s$, and the nodes $1, \ldots, n$. In this network, there are arcs $(q,i)$ and $(i,s)$ for all $i = 1, \ldots, n$, as well as arcs $(i,j)$ if $\alpha(i,j) = 3$. Note, $\tilde{w}_{q,i}$ and $\tilde{w}_{i,j}$ are the costs of the respective arcs. It is easily seen that we can compute an optimal solution of $\mathcal{L}_2$ by solving a classical min cost flow problem in a network which results from slightly adapting $\tilde{\mathcal{N}}$.

After all, we compute a lower bound on the optimal value $z^*_{\mathfrak{s}}$ of the currently processed subproblem $\mathfrak{P}_{\mathfrak{s}}$ by solving the Lagrangian dual with a bundle method, see the next chapter for details.

In order to obtain an upper bound, we take the solution of the inner problem that – after terminating the bundle method – produced the best lower bound. We round down the values of the variables $y_{i,j}$ with $\alpha(i,j) = 2$ as little as possible until the values of these variables allow feasible solutions of $\mathfrak{P}_s$. Then, we determine a feasible coloring applying PACK LONGEST which starts with a pre-packing that is given by the remaining $y_{i,j} = 1$ with $\alpha(i,j) = 2$. As in the LP based implementation, we additionally compute a sequence of feasible colorings and corresponding upper bounds by iteratively executing an adapted Algorithm 11, see page 97.

**Branching and Node Selection Policy** These are the same as for the LP based NETWORK branch-and-bound, see page 98.

In Section 6.1, we discuss the computational results of the above-mentioned exact solution approaches for both the ASSIGNMENT BP and the NETWORK BP.

CHAPTER 6

# Computational Results

THIS chapter gathers our computational experience of solving various versions of SRS. The results for the particular real-world optimization problem at BASF in Ludwigshafen are presented in Section 6.2. In the corresponding versions, shunting over a hump is an important issue. Section 6.1 deals with versions that answer the question of how the required rearrangements at BASF can be performed efficiently in case we do not allow additional shunting moves over the hump. All results discussed in this chapter were obtained for real input data. In particular, the instances correspond to daily incoming sequences of railcars at BASF. The presented heuristics and branch-and-bound procedures are implemented in the programming language C, and all solution methods were run on an Intel Quad Core with 3.2 GHz and 12 GB RAM.

## 6.1 Versions with no Shunting

In the following, we represent the results for five $t$**-minimizing**, **unbounded**, **no shunting**, $g$**-blocks** versions that are equivalent to MVC of polygon-circle graphs, see Subsection 4.3.1. At the same time, we evaluate the solution methods – described in Chapter 5 – for solving MVC of the corresponding polygon-circle graphs.

In Table 6.1, each line corresponds to a practical SRS instance. The first column lists the total number of incoming railcars, the second column contains the number of different groups within the incoming sequence. Note that for all **unbounded**, $g$**-blocks** versions, there is an optimal schedule in which any two consecutively incoming railcars $i$

and $i+1$ that belong to the same group are assigned identical moves. As a consequence, it is sensible to perform the following preprocessing for **unbounded**, $g$**-blocks** versions: in the given input sequence shrink any consecutive block of railcars that belong to the same group to a single railcar of this group.

For each of the considered versions, in order to obtain a good feasible – possibly optimal – schedule for a given input sequence we execute the following steps:

- we preprocess the input sequence as described above,
- we construct a particular polygon-circle graph – according to the version and to the preprocessed input sequence – see the proofs of the respective theorems in Subsection 4.3.1,
- we preprocess the resulting polygon-circle graph, see Section 5.2,
- we determine a good feasible – possibly optimal – coloring of the preprocessed polygon-circle graph,
- and we translate this coloring into a feasible schedule for rearranging the railcars – the number of colors corresponds to the number of occupied tracks.

For the rest of this section, we discuss the results in terms of the coloring of the preprocessed polygon-circle graphs.

First of all, let us empirically evaluate the quality of the heuristically computed colorings. In Table 6.1, there are five columns for each version that list: the max clique size $\omega$, the chromatic number $\chi$, and the difference between $\chi$ and the number of colors used in the colorings computed by the heuristics PACK LONGEST (PL), MAXIS (MIS), and FIRST FIT (FF). An empty cell means that the coloring obtained by the respective method is optimal. Note that the heuristics run in less than a second for each of the given graphs.

On average, PACK LONGEST produces colorings with 0.55 colors more than the optimal coloring. For MAXIS and FIRST FIT, these average gaps read 0.96 and 1.32, respectively. The worst-case optimality gap of the best heuristical coloring is 40 percent. Only in 7 percent of the preprocessed polygon-circle graphs, PACK LONGEST is outperformed by one of the other heuristics. For two out of three graphs, the best heuristical coloring can immediately be shown to be optimal,

since the number of colors used equals $\omega$. Remember, $\omega$ is easily computable by Algorithm 1. For the other graphs, the corresponding cells are highlighted in gray. In 80 percent of the instances for the **split** versions, we are not aware of a proven optimal coloring after computing the heuristical colorings and $\omega$. For the 0-**split** versions, this is the case for only one instance. A reason is that the given polygon-circle graph for a 0-**split** version is usually much smaller than the graph of the corresponding **split** version.

In the following, let us consider the preprocessed polygon-circle graphs for which the best heuristical coloring is not immediately proven to be optimal. For these graphs, we discuss the computational times of the exact solution approaches that are based on either the ASSIGNMENT BP or the NETWORK BP. For both BP formulations and for all instances, $\mathfrak{C}$ could be generated entirely within 5 seconds. That is, the ASSIGNMENT BPs are constructed according to (5.1), (5.2), (5.7), (5.4), and (5.5); and the NETWORK BPs are generated in accordance with (5.18)–(5.25).

The results for the methods tailored to the ASSIGNMENT BP are shown in the Tables 6.2 and 6.3; Table 6.2 is for the polygon-circle graphs that relate to instances of version $\underline{t}$-**st,ub**|**nsh,co,sp**|**or,**$g$-**bl**, Table 6.3 is for version $\underline{t}$-**st,ub**|**nsh,tw,sp**|**or,**$g$-**bl**. The Tables 6.4 and 6.5 illustrate the results for the NETWORK BP approaches; Table 6.4 is for the instances of version $\underline{t}$-**st,ub**|**nsh,co,sp**|**or,**$g$-**bl**, Table 6.5 is for version $\underline{t}$-**st,ub**|**nsh,tw,sp**|**or,**$g$-**bl**. In the tables, each row contains the figures for one polygon-circle graph; the number of nodes and the number of arcs are listed in the third and fourth column. Note that the preprocessed polygon-circle graph for a **concurrent** version is mostly smaller than the preprocessed graph of the corresponding **time windows** version for the same SRS instance.

**MIP solver** The sixth till ninth column of the Tables 6.2–6.5 illustrate the performance of the commercial MIP solvers CPLEX 12.1 and Gurobi 3.0.0 for solving both the ASSIGNMENT BPs and the NETWORK BPs with default settings. Note, for the NETWORK BP, telling the MIP solvers that the objective value is integral improves their performance. We provided both solvers with the best heuristical coloring as a feasible start solution.

| instances | | $\underline{t}$-st,ub\|nsh,tw,0-sp\|or,$g$-bl | | | | | $\underline{t}$-st,ub\|nsh,co,0-sp\|or,$g$-bl | | | | | $\underline{t}$-st,ub\|nsh,co,0-sp\|fr,$g$-bl | | | | | $\underline{t}$-st,ub\|nsh,tw,sp\|or,$g$-bl | | | | | $\underline{t}$-st,ub\|nsh,co,sp\|or,$g$-bl | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cars | groups | $\omega$ | $\chi$ | PL | MIS | FF | $\omega$ | $\chi$ | PL | MIS | FF | $\omega$ | $\chi$ | PL | MIS | FF | $\omega$ | $\chi$ | PL | MIS | FF | $\omega$ | $\chi$ | PL | MIS | FF |
| 126 | 45 | 12 | 12 | | | | 11 | 11 | | | | 9 | 9 | | | | 9 | 9 | | +2 | +1 | 8 | 8 | **+1** | +1 | +2 |
| 140 | 45 | 12 | 12 | | | | 10 | 10 | | | | 8 | 8 | | | | 11 | 11 | | | | 9 | 9 | | | |
| 207 | 57 | 16 | 16 | | | +2 | 14 | 14 | | +1 | +1 | 12 | 12 | | | | 13 | 13 | | +2 | +1 | *9* | *10* | +2 | **+1** | +2 |
| 209 | 51 | 16 | 16 | | | | 15 | 15 | | | | 14 | 14 | | | | 12 | 12 | **+1** | +1 | +3 | 12 | 12 | +1 | | +2 |
| 222 | 61 | 21 | 21 | | | | 20 | 20 | | | | 14 | 14 | | | | 13 | 13 | **+1** | +1 | +3 | 13 | 13 | | | +3 |
| 244 | 62 | 26 | 26 | | | | 26 | 26 | | | | 20 | 20 | | | | 12 | 12 | **+3** | +4 | +5 | 12 | 12 | **+1** | +3 | +3 |
| 245 | 61 | 21 | 21 | | | | 21 | 21 | | | | 15 | 15 | | | | *12* | *13* | **+2** | +3 | +3 | 12 | 12 | **+1** | +2 | +5 |
| 255 | 53 | 18 | 18 | | | | 16 | 16 | | | | 15 | 15 | | | | *10* | *11* | **+2** | +5 | +7 | *10* | *11* | **+1** | +3 | +5 |
| 269 | 65 | 21 | 21 | | +1 | +1 | 20 | 20 | | | | 17 | 17 | | | | 13 | 13 | **+1** | +3 | +4 | 13 | 13 | **+1** | +2 | +4 |
| 272 | 64 | 25 | 25 | | | | 24 | 24 | | | | 21 | 21 | | | | *12* | *13* | **+2** | +4 | +4 | 11 | 11 | **+2** | +2 | +3 |
| 280 | 62 | 25 | 25 | | | | 24 | 24 | | | | 20 | 20 | | | | 14 | 14 | **+1** | +2 | +3 | 13 | 13 | **+1** | +2 | +2 |
| 287 | 65 | 24 | 24 | | | | 24 | 24 | | | | 19 | 19 | | | | 14 | 14 | **+1** | +2 | +1 | 14 | 14 | | +1 | +1 |
| 289 | 67 | 23 | 23 | | | | 22 | 22 | | | | 20 | 20 | | | | 13 | 13 | **+1** | +4 | +4 | 13 | 13 | +2 | **+1** | +2 |
| 304 | 71 | 25 | 25 | | | | 23 | 23 | | | | 21 | 21 | | | | *14* | *15* | **+2** | +2 | +6 | 14 | 14 | **+3** | +3 | +4 |
| 306 | 67 | 22 | 22 | | | | 20 | 20 | | | | 16 | 16 | | | | 13 | 13 | **+2** | +4 | +7 | 12 | 12 | **+1** | +3 | +2 |
| 311 | 70 | 25 | 25 | | | | 24 | 24 | | | | 20 | 20 | +1 | +1 | | 14 | 14 | +3 | **+2** | +4 | 14 | 14 | +2 | +3 | **+1** |
| 317 | 69 | 25 | 25 | | | | 25 | 25 | | | | 19 | 19 | **+1** | +1 | +1 | 15 | 15 | **+1** | +2 | +2 | 15 | 15 | +2 | +4 | |
| 319 | 66 | 28 | 28 | | | | 23 | 23 | | | | 21 | 21 | | | | 16 | 16 | **+1** | +4 | +8 | *14* | *15* | **+1** | +3 | +6 |
| 343 | 70 | 26 | 26 | | | | 24 | 24 | | | | 21 | 21 | | | | 15 | 15 | **+1** | +2 | +5 | 15 | 15 | **+1** | +2 | +4 |
| 365 | 71 | 24 | 24 | | | | 21 | 21 | | | | 17 | 17 | +1 | | | 13 | 13 | **+3** | +4 | +5 | 12 | 12 | **+1** | +3 | +2 |

**Table 6.1.** Max clique size, chromatic number, and values of heuristical solutions for our real-world instances

| real-world instances | | preprocessed polygon-circle graphs | | ASSIGNMENT BP | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | MIP solver | | | | LP based branch-and-bound | | | | | | |
| | | | | | Gurobi 3.0.0 | | CPLEX 12.1.0 | | Gurobi 3.0.0 | | CPLEX 12.1.0 | | lp_solve 5.5.0.15 | |
| cars | groups | nodes | edges | binary variables | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes |
| 126 | 45 | 29 | 216 | 128 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 207 | 57 | 48 | 519 | 310 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 14 |
| 244 | 62 | 67 | 1091 | 481 | 1 | 0 | 1 | 0 | 2 | 0 | 2 | 0 | 3 | 1 |
| 245 | 61 | 72 | 1197 | 436 | 1 | 0 | 1 | 0 | 2 | 0 | 2 | 1 | 1 | 0 |
| 255 | 53 | 56 | 772 | 342 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 269 | 65 | 87 | 1462 | 629 | 1 | 0 | 1 | 0 | 2 | 0 | 2 | 1 | 3 | 0 |
| 272 | 64 | 88 | 1671 | 618 | 1 | 0 | 1 | 0 | 3 | 0 | 4 | 0 | 4 | 0 |
| 280 | 62 | 84 | 1692 | 481 | 1 | 0 | 1 | 0 | 5 | 1 | 4 | 16 | 5 | 7 |
| 289 | 67 | 58 | 863 | 364 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 304 | 71 | 99 | 2373 | 872 | 1 | 0 | 1 | 0 | 13 | 27 | 12 | 0 | 17 | 1 |
| 306 | 67 | 77 | 1237 | 465 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 2 | 1 |
| 311 | 70 | 83 | 1590 | 532 | 1 | 0 | 1 | 0 | 4 | 0 | 4 | 0 | 3 | 0 |
| 319 | 66 | 107 | 2686 | 861 | 1 | 0 | 3 | 0 | 18 | 0 | 20 | 15 | 60 | 33 |
| 343 | 70 | 93 | 2040 | 705 | 1 | 0 | 1 | 0 | 6 | 4 | 6 | 0 | 17 | 12 |
| 365 | 71 | 81 | 1425 | 508 | 1 | 0 | 1 | 0 | 3 | 1 | 3 | 0 | 4 | 8 |

**Table 6.2.** Computational times for our real-world instances of version $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl** regarding the ASSIGNMENT BP approaches

| real-world instances | | preprocessed polygon-circle graphs | | ASSIGNMENT BP | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | MIP solver | | | | LP based branch-and-bound | | | | | | |
| | | | | | Gurobi 3.0.0 | | CPLEX 12.1.0 | | Gurobi 3.0.0 | | CPLEX 12.1.0 | | lp_solve 5.5.0.15 | |
| cars | groups | nodes | edges | binary variables | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes |
| 209 | 51 | 74 | 1113 | 502 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 3 | 2 | 0 |
| 222 | 61 | 69 | 1104 | 438 | 1 | 0 | 1 | 0 | 2 | 0 | 2 | 0 | 1 | 3 |
| 244 | 62 | 112 | 2286 | 1097 | 2 | 0 | 2 | 0 | 7 | 0 | 19 | 7 | 76 | 16 |
| 245 | 61 | 69 | 1129 | 525 | 1 | 0 | 1 | 0 | 1 | 4 | 2 | 0 | 2 | 0 |
| 255 | 53 | 92 | 1534 | 804 | 1 | 0 | 2 | 0 | 5 | 14 | 4 | 16 | 25 | 28 |
| 269 | 65 | 84 | 1470 | 585 | 1 | 0 | 1 | 0 | 2 | 0 | 2 | 1 | 3 | 0 |
| 272 | 64 | 100 | 2151 | 831 | 1 | 0 | 1 | 0 | 8 | 8 | 10 | 33 | 29 | 54 |
| 280 | 62 | 93 | 2172 | 561 | 1 | 0 | 1 | 0 | 8 | 0 | 8 | 0 | 10 | 9 |
| 287 | 65 | 96 | 1904 | 682 | 1 | 0 | 1 | 0 | 5 | 0 | 5 | 0 | 5 | 0 |
| 289 | 67 | 110 | 2187 | 866 | 1 | 0 | 1 | 0 | 7 | 3 | 6 | 1 | 15 | 4 |
| 304 | 71 | 127 | 3209 | 1224 | 2 | 0 | 6 | 0 | 19 | 6 | 21 | 10 | 85 | 84 |
| 306 | 67 | 94 | 1765 | 756 | 1 | 0 | 1 | 0 | 3 | 0 | 4 | 1 | 5 | 0 |
| 311 | 70 | 102 | 2327 | 780 | 1 | 0 | 1 | 0 | 9 | 0 | 9 | 5 | 12 | 5 |
| 317 | 69 | 111 | 2247 | 896 | 1 | 0 | 1 | 0 | 5 | 0 | 5 | 1 | 6 | 0 |
| 319 | 66 | 137 | 3942 | 1321 | 2 | 0 | 84 | 186 | 59 | 8 | 89 | 11 | 120 | 15 |
| 343 | 70 | 96 | 2235 | 700 | 1 | 0 | 1 | 0 | 9 | 1 | 9 | 1 | 11 | 1 |
| 365 | 71 | 112 | 2386 | 1080 | 1 | 0 | 1 | 0 | 8 | 1 | 14 | 10 | 97 | 43 |

**Table 6.3.** Computational times for our real-world instances of version $\underline{t}$**-st,ub|nsh,tw,sp|or,**$g$**-bl** regarding the ASSIGNMENT BP approaches

| real-world instances | | preprocessed polygon-circle graphs | | | NETWORK BP | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | MIP solver | | | | LP based branch-and-bound | | | | | | Lagrangian branch-and-bound | |
| | | | | | Gurobi 3.0.0 | | CPLEX 12.1.0 | | Gurobi 3.0.0 | | CPLEX 12.1.0 | | lp_solve 5.5.0.15 | | | |
| cars | groups | nodes | edges | binary variables | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes |
| 126 | 45 | 29 | 216 | 121 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 |
| 207 | 57 | 48 | 519 | 433 | 1 | 8 | 1 | 0 | 1 | 28 | 1 | 28 | 6 | 28 | 10 | 10 |
| 244 | 62 | 67 | 1091 | 857 | 2 | 0 | 2 | 0 | 2 | 0 | 2 | 0 | 239 | 24 | 162 | 55 |
| 245 | 61 | 72 | 1197 | 833 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 255 | 53 | 56 | 772 | 486 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 22 | 2 | 4 | 4 |
| 269 | 65 | 87 | 1462 | 1301 | 2 | 0 | 2 | 0 | 2 | 0 | 2 | 0 | 43 | 2 | 15 | 2 |
| 272 | 64 | 88 | 1671 | 1375 | 3 | 0 | 3 | 0 | 3 | 0 | 4 | 2 | 42 | 2 | 15 | 2 |
| 280 | 62 | 84 | 1692 | 1197 | 2 | 0 | 1 | 0 | 5 | 10 | 3 | 2 | 331 | 26 | 91 | 26 |
| 289 | 67 | 58 | 863 | 485 | 1 | 0 | 1 | 0 | 1 | 3 | 1 | 1 | 1 | 0 | 200 | 530 |
| 304 | 71 | 99 | 2373 | 1782 | 11 | 0 | 14 | 0 | 18 | 2 | 24 | 21 | $>600^{+1}$ | 44 | $>600^{+1}$ | 74 |
| 306 | 67 | 77 | 1237 | 1071 | 1 | 0 | 2 | 0 | 2 | 2 | 2 | 2 | 41 | 2 | 11 | 2 |
| 311 | 70 | 83 | 1590 | 1328 | 2 | 0 | 1 | 0 | 2 | 0 | 3 | 2 | 103 | 5 | 21 | 5 |
| 319 | 66 | 107 | 2686 | 2251 | 66 | 10 | $>600^{=}$ | 211 | $>600^{=}$ | 144 | $>600^{=}$ | 198 | $>600^{=}$ | 35 | $>600^{=}$ | 40 |
| 343 | 70 | 93 | 2040 | 1312 | 3 | 0 | 2 | 0 | 4 | 2 | 3 | 4 | $>600^{+1}$ | 52 | 261 | 70 |
| 365 | 71 | 81 | 1425 | 1262 | 2 | 0 | 2 | 0 | 3 | 6 | 4 | 6 | 209 | 20 | 180 | 50 |

**Table 6.4.** Computational times for our real-world instances of version $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl** regarding the NETWORK BP approaches

| real-world instances | | preprocessed polygon-circle graphs | | | NETWORK BP | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | MIP solver | | | | LP based branch-and-bound | | | | | | Lagrangian branch-and-bound | |
| | | | | | Gurobi 3.0.0 | | CPLEX 12.1.0 | | Gurobi 3.0.0 | | CPLEX 12.1.0 | | lp_solve 5.5.0.15 | | | |
| cars | groups | nodes | edges | binary variables | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes | time in sec | B&B nodes |
| 209 | 51 | 74 | 1113 | 1155 | 2 | 0 | 2 | 0 | 2 | 3 | 1 | 0 | 62 | 3 | 20 | 3 |
| 222 | 61 | 69 | 1104 | 951 | 2 | 0 | 1 | 0 | 1 | 2 | 2 | 0 | 62 | 3 | 14 | 3 |
| 244 | 62 | 112 | 2286 | 2408 | 10 | 0 | 8 | 0 | 69 | 15 | 17 | 3 | >600 $^{+1}$ | 38 | >600 $^{+1}$ | 49 |
| 245 | 61 | 69 | 1129 | 759 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 44 | 4 | 35 | 20 |
| 255 | 53 | 92 | 1534 | 1552 | 4 | 0 | 6 | 0 | 4 | 0 | 13 | 10 | >600 $^{+1}$ | 46 | >600 $^{+1}$ | 111 |
| 269 | 65 | 84 | 1470 | 1090 | 3 | 0 | 2 | 0 | 6 | 20 | 2 | 1 | >600 $^{+1}$ | 49 | >600 $^{+1}$ | 195 |
| 272 | 64 | 100 | 2151 | 1762 | 8 | 0 | 15 | 0 | 18 | 6 | 8 | 0 | >600 $^{+1}$ | 50 | >600 $^{+1}$ | 89 |
| 280 | 62 | 93 | 2172 | 1460 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 |
| 287 | 65 | 96 | 1904 | 1683 | 3 | 0 | 2 | 0 | 3 | 0 | 3 | 0 | 4 | 0 | 3 | 0 |
| 289 | 67 | 110 | 2187 | 2082 | 7 | 0 | 7 | 0 | 6 | 0 | 6 | 0 | 5 | 0 | 6 | 0 |
| 304 | 71 | 127 | 3209 | 3406 | 54 | 0 | 385 | 122 | >600 $^{+1}$ | 103 | >600 $^{=}$ | 219 | >600 $^{+1}$ | 28 | >600 $^{+1}$ | 23 |
| 306 | 67 | 94 | 1765 | 1565 | 5 | 0 | 4 | 0 | 17 | 13 | 5 | 1 | >600 $^{+1}$ | 41 | 488 | 77 |
| 311 | 70 | 102 | 2327 | 1832 | 5 | 0 | 5 | 0 | 10 | 4 | 10 | 10 | 405 | 27 | 204 | 27 |
| 317 | 69 | 111 | 2247 | 2464 | 8 | 0 | 7 | 0 | 8 | 0 | 8 | 0 | 8 | 0 | 8 | 0 |
| 319 | 66 | 137 | 3942 | 3672 | 65 | 0 | 348 | 0 | 275 | 11 | 118 | 1 | >600 $^{+1}$ | 27 | >600 $^{+1}$ | 13 |
| 343 | 70 | 96 | 2235 | 1444 | 3 | 0 | 3 | 0 | 4 | 0 | 4 | 2 | >600 $^{+1}$ | 47 | >600 $^{+1}$ | 142 |
| 365 | 71 | 112 | 2386 | 2507 | 15 | 0 | 21 | 0 | 25 | 2 | 53 | 21 | >600 $^{+1}$ | 43 | >600 $^{+1}$ | 43 |

**Table 6.5.** Computational times for our real-world instances of version $\underline{t}$**-st,ub|nsh,tw,sp|or,**$g$**-bl** regarding the NETWORK BP approaches

**LP based branch-and-bound** Further columns of the tables list the results of the LP based branch-and-bound procedures for both BP formulations, see Subsection 5.4.1 and 5.4.2 for a description of these methods. We tested three LP solvers: CPLEX 12.1, Gurobi 3.0.0, and lp_solve 5.5.0.15. The free lp_solve was developed by Michel Berkelaar, Kjell Eikland, and Peter Notebaert, and it is under the GNU Lesser General Public License. In any of these implementations, we perform ITERATIVE ROUNDING for every branch-and-bound node in order to obtain feasible colorings and corresponding upper bounds. On average, the best results were achieved with $\delta = 0.1$.

**Lagrangian branch-and-bound** The last two columns of the Tables 6.4 and 6.5 refer to the Lagrangian branch-and-bound – which is described in Subsection 5.4.2 – for solving the NETWORK BP. In order to solve the Lagrangian dual, we apply the bundle method that is implemented in the free ConicBundle 0.3.2 library. ConicBundle is being developed by Christoph Helmberg, and it is under the GNU General Public License. Remember, in the inner problem of the Lagrangian dual we need to solve a min cost flow problem. Here we use MCF 1.3 which is a network simplex implementation introduced and coded by Andreas Löbel. MCF is under ZIB Academic License.

For each run of any solution approach, we impose a time limit of 10 minutes. In case no proven optimal coloring is produced within the time limit for a particular instance, the additive gap between the number of colors of the best coloring so far and the chromatic number is written superscript within the corresponding cell. Now, let us discuss the computational results for the exact solution approaches.

### ASSIGNMENT BP

Gurobi solves each ASSIGNMENT BP of the **concurrent** and **time windows** versions within 2 seconds at the root node. Except two instances for the **time windows** version, this is the same with CPLEX. The worst performance of CPLEX is 84 seconds for 186 branch-and-bound nodes. With the LP based branch-and-bound implementations we solved all instances within 2 minutes. The two commercial LP solvers Gurobi and CPLEX almost have the same impact on the performance of the branch-and-bound. The computations with lp_solve as LP solver take only a bit longer.

### NETWORK BP

Any NETWORK BP of the **concurrent** and **time windows** versions is solved by Gurobi within roughly a minute. For most instances, CPLEX also succeeds after a few seconds. Only for three graphs, the computation takes longer than 5 minutes and for one of these instances CPLEX has found an optimal coloring after 10 minutes but has not yet proven that it is optimal. In 30 of 32 instances, a verified optimal coloring is found at the root node by both solvers.

The LP based branch-and-bound with CPLEX as LP solver produced optimal colorings for all graphs within 10 minutes, mostly within a few seconds. However, for two graphs, the optimality of the determined coloring could not be verified. On average, this implementation with CPLEX slightly outperforms the branch-and-bound with Gurobi as LP solver. In case of lp_solve, verified optimal colorings are obtained for 20 of 32 instances before the time limit of ten minutes is reached. For the remaining instances, the best coloring found has at most one color more than the optimal coloring. The Lagrangian branch-and-bound produces proven optimal colorings for 22 graphs and the worst optimality gap to $\chi$ is also one color. Considering our implementations which do not incorporate any commercial solvers, the Lagrangian approach yields the best results.

After all, we compare the approaches for the ASSIGNMENT BP with those for the NETWORK BP. With a few exceptions, the MIP solvers solve both kind of BPs very fast. The NETWORK BP seems to be less affected by symmetry than the ASSIGNMENT BP. Nevertheless, the LP based branch-and-bound for the ASSIGNMENT BP clearly outperforms the LP based approach for the NETWORK BP. This could be reasoned by the fact that the NETWORK BPs on average have twice as many binary variables as the corresponding ASSIGNMENT BPs, see the fifth column of the Tables 6.2 – 6.5.

Of course, the proposed preprocessing is essential for fast computations. For example, we also tested the performance of the MIP solvers Gurobi and CPLEX when solving the BP models that we obtain from a given SRS instance without any preprocessing. That is, we do not shrink the given input sequence as well as the corresponding polygon-circle graph. Moreover, we do not provide the BP model with a maximum clique of the resulting graph, and no non-trivial clique constraints are included. For any of the resulting ASSIGNMENT or NETWORK BP

models, neither of the two solvers Gurobi and CPLEX could compute an optimal coloring within one hour, and the best coloring found after that time is worse than the best coloring obtained by the presented heuristics.

In particular, the clique constraints – (5.7) or (5.19) – of the BP models strongly affect the performance of the solution approaches. Without these clique constraints the computational times increase substantially. Due to time and space constraints, it is impossible to add arbitrarily many of these inequalities to the BP models a priori. In order to solve larger instances than our practical ones, we could alternatively separate the clique constraints in our LP based approaches. For both BP formulations, a violated clique constraint can be found by applying Algorithm 1 for determining an MWC of particular polygon-circle graphs whose vertices are weighted according to particular values of the current LP solution.

Let us conclude this section with the following bottom line. We can determine optimal schedules for all of our practical SRS instances quickly, even with an implementation that does not include any commercial software.

## 6.2 Versions with real Application at a Hump Yard

This section deals with the $\mathcal{NP}$-hard real-world optimization problem $t$**-st,**$b$**-bd**$|\underline{h}$**,**$\underline{r}$**-hsh,co,sp**$|o$**-or,**$g$**-bl** at BASF, see Section 4.4 for a detailed description. We present a very fast heuristical method and a fast exact approach. In particular, we discuss results for real data from theoretical as well as from practical point of view.

### Solution Approaches

We start with our heuristical approach. The basic idea is to transform an optimal schedule of the respective **unbounded** version into a schedule which complies with the track lengths. Note, for the **unbounded** case, Algorithm 8 yields a schedule with minimum number of humping steps and – sticking to this number – with the least number of railcar moves at once. Tables 6.6 and 6.7 present results for practical instances. The third column contains the minimum number $h^*_{\text{ub}}$ of humping steps in case of **unbounded** tracks. The next three columns list the

number of railcars moved during the respective humping step, that is, the number of railcars parked on the track which is immediately emptied before the respective humping step. Since at most 30 railcars can be placed on a sorting track in the rail yard at BASF – that is, $b = 30$ – these schedules are obviously infeasible for the actual **bounded** case. However, by distributing the railcars assigned to an overfilled track to additionally available free tracks we easily get a feasible schedule. Of course, the number of humping steps increases, while the number of railcar moves stays the same. In Tables 6.6 and 6.7 the seventh and ninth column list the number $\bar{h}$ of humping steps and the number $\bar{r}$ of railcar moves of the computed schedules, respectively.

For computing optimal schedules of the actual optimization problem – that is, the $b$-**bounded** version – we propose the following three-phase approach.

In the *first phase*, we determine a feasible schedule with the above heuristical approach to get bounds for the optimal objective values. Of course, $\bar{h}$ and $\bar{r}$ are upper bounds for the minimum number $h^*$ of humping steps and the corresponding minimum number $r^*$ of railcar moves, respectively. Let $\tilde{n}_1$ be the number of railcars which can directly move from the input track to an output track – that is, $\tilde{n}_1 = |S_1|$ – see Algorithm 8. Note that the output tracks are assumed to be long enough for the entire outbound trains. Then, $n - \tilde{n}_1$ gives a lower bound on $r^*$, and $\underline{h} := \max\{(n - \tilde{n}_1)/b, h^*_{\text{ub}}\}$ provides a lower bound on $h^*$. For our practical instances, these lower bounds are shown in the eleventh and twelfth column of the Tables 6.6 and 6.7.

In the *second phase*, we determine the minimum number $h^*$ of humping steps. If the lower bound on $h^*$ computed in phase 1 equals the upper bound $\bar{h}$, it follows $h^* = \bar{h}$, and we continue with phase 3. This is the case for roughly 40 percent of our instances. Otherwise, we determine $h^*$ by solving the Binary Programm (6.1)–(6.10) – see next page – which extends the idea of Algorithm 8 to the $b$-**bounded** case. Actually, we are not interested in the solution itself, only in the optimal value $h^*$.

In this formulation, $P_h^{\mathcal{E}^c}$ denotes the set of paths which "*use*" the $h$-th humping step regarding the cyclic track execution order $\mathcal{E}^c$ for $t$ tracks with $\hat{h}$ humping steps. The decision variables read $x_{i,p} = 1$ if and only if the $i$-th railcar takes path $p$, $y_p = 1$ if and only if there is a railcar taking path $p$, and $z_h = 1$ if and only if humping step $h$ is executed.

$$\min \sum_{h} z_h \tag{6.1}$$

$$\sum_{p} x_{i,p} = 1 \qquad \forall i \tag{6.2}$$

$$x_{i,p} + \sum_{\hat{p} \geq p} x_{j,\hat{p}} \leq 1 \qquad \forall p, \forall i < j \text{ with } 1 + \sum_{m=1}^{l-1} g_m \leq s_j < s_i \leq \sum_{m=1}^{l} g_m \text{ for an } l \in \{1, \ldots, o\} \tag{6.3}$$

$$x_{j,p} + \sum_{\hat{p} \geq p+1} x_{i,\hat{p}} \leq 1 \qquad \forall p, \forall i < j \text{ with } 1 + \sum_{m=1}^{l-1} g_m \leq s_i < s_j \leq \sum_{m=1}^{l} g_m \text{ for an } l \in \{1, \ldots, o\} \tag{6.4}$$

$$\sum_{i} \sum_{p \in P_h^{\mathcal{E}^c}} x_{i,p} \leq b \qquad \forall h \tag{6.5}$$

$$y_p - x_{i,p} \geq 0 \qquad \forall p, \forall i \tag{6.6}$$

$$y_p - y_{p+1} \geq 0 \qquad \forall p < f(t, \hat{h}) \tag{6.7}$$

$$z_h - y_p \geq 0 \qquad \forall h, \forall p \in P_h^{\mathcal{E}^c} \tag{6.8}$$

$$\sum_{h} z_h \geq \underline{h} \tag{6.9}$$

$$x_{i,p},\ y_p,\ z_h \in \{0,1\} \qquad 1 \leq i \leq n,\ 1 \leq p \leq f(t, \hat{h}),\ 1 \leq h \leq \hat{h} \tag{6.10}$$

Restriction (6.2) ensures that each railcar takes exactly one path; (6.3) and (6.4) yield the desired structure **ordered blocks** of the outbound trains; (6.5) considers the track length; due to (6.7) only the "*first*" paths are chosen; (6.9) provides the lower bound on $h^*$; (6.6) and (6.8) are coupling constraints. The objective is to minimize $h^*$.

It is easily seen that solving the BP (6.1)–(6.10) with $\hat{h} = \bar{h}$ yields an optimal solution of version $t$**-st,**$b$**-bd**$|\underline{h}$**-hsh,co,sp**$|o$**-or,**$g$**-bl**. How-

ever, as already mentioned we are at this point only interested in the number $h^*$. Thus, we solve the Binary Programm with $\hat{h} = \bar{h} - 1$. In case of infeasibility, we know $h^* = \bar{h}$; otherwise, the optimal value provides $h^*$.

In Tables 6.6 and 6.7, the columns 13 – 15 list $h^*$ as well as the computational times for solving the Binary Programm via the solvers CPLEX 12.1 and Gurobi 3.0.0, respectively.

Finally, in the *third phase*, we compute a schedule with minimum number $r^*$ of railcar moves – performing the minimum number $h^*$ of humping steps – by solving a second Binary Program which reads as follows.

$$\min \sum_{h} \sum_{i} \sum_{p \in P_h^{\mathcal{E}^c}} x_{i,p} \tag{6.11}$$

subject to

$$\sum_{h} z_h = h^* \qquad \forall h \tag{6.12}$$

as well as to (6.2) – (6.8) and (6.10), where $\hat{h} = h^*$.

Of course, it is possible to merge phases 2 and 3, that is, to determine an optimal schedule by solving a single Binary Program with an weighted objective function. However, that turned out to be less efficient regarding the overall computational time.

### Computational Results

For our instances, the minimum number $r^*$ of railcar moves as well as the computational times for solving the second Binary Programm via the solvers CPLEX 12.1 and Gurobi 3.0.0 are contained in the columns 16 – 18 of the Tables 6.6 and 6.7.

At first, let us compare the computational times of the solution methods. The heuristically computed schedules are obtainable very fast within 1 second. The computational times for the $h$-**minimizing** Binary Programm vary between 1 second and 7 minutes when solved by Gurobi 3.0.0 and between 1 second and 27 minutes applying CPLEX 12.1; for both solvers with default parameter setting. At least for the four instances which could not be solved within one minute,

| real-world instances | | **unbounded case** optimal solution (by greedy in 1 sec) | | | | **bounded case** ($b = 30$) heuristical solution | | | | lower bounds | | optimal solutions (via MIP-solver) minimize $h$ | | | minimize $r$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | railcar moves at humping step | | | | | | | | | | time in sec | | | time in sec | |
| cars | groups | $h^*_{\text{ub}}$ | 1 | 2 | 3 | $\bar{h}$ | $\Delta_h$ | $\bar{r}$ | $\Delta_r$ | on $r^*$ | on $h^*$ | $h^*$ | Gurobi 3.0.0 | CPLEX 12.1 | $r^*$ | Gurobi 3.0.0 | CPLEX 12.1 |
| 126 | 45 | 2 | 47 | 3 | | 3 | 1 | 50 | 0 | 50 | 2 | 2 | 1 | 1 | 50 | 1 | 1 |
| 140 | 45 | 2 | 25 | 18 | | 2 | 0 | 43 | 0 | 43 | 2 | 2 | | | 43 | 1 | 1 |
| 207 | 57 | 2 | 70 | 22 | | 4 | 1 | 92 | 7 | 85 | 3 | 3 | 1 | 1 | 85 | 1 | 1 |
| 209 | 51 | 2 | 54 | 15 | | 3 | 0 | 69 | 0 | 69 | 3 | 3 | | | 69 | 1 | 1 |
| 222 | 61 | 2 | 76 | 26 | | 4 | 0 | 102 | 2 | 100 | 4 | 4 | | | 100 | 1 | 1 |
| 244 | 62 | 3 | 83 | 31 | 1 | 6 | 2 | 115 | 9 | 106 | 4 | 4 | 6 | 2 | 106 | 1 | 1 |
| 245 | 61 | 2 | 76 | 25 | | 4 | 0 | 101 | 6 | 95 | 4 | 4 | | | 95 | 1 | 1 |
| 255 | 53 | 2 | 57 | 29 | | 3 | 0 | 86 | 8 | 78 | 3 | 3 | | | 78 | 1 | 1 |
| 269 | 65 | 2 | 91 | 14 | | 5 | 1 | 105 | 0 | 105 | 4 | 4 | 1 | 1 | 105 | 1 | 1 |
| 272 | 64 | 2 | 78 | 21 | | 4 | 0 | 99 | 3 | 96 | 4 | 4 | | | 96 | 1 | 1 |

**Table 6.6.** Real-world results for versions $t$**-st,**$\cdot|\underline{h}$**,**$\underline{r}$**-hsh,co,sp**$|o$**-or,**$g$**-bl** – Part I

| real-world instances | | **unbounded case** optimal solution (by greedy in 1 sec) | | | | **bounded case** ($b = 30$) heuristical solution | | | | | | optimal solutions (via MIP-solver) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | railcar moves at humping step | | | | | | | lower bounds | | minimize $h$ time in sec | | | minimize $r$ time in sec | | |
| cars | groups | $h^*_{\text{ub}}$ | 1 | 2 | 3 | $\bar{h}$ | $\Delta_h$ | $\bar{r}$ | $\Delta_r$ | on $r^*$ | on $h^*$ | $h^*$ | Gurobi 3.0.0 | CPLEX 12.1 | $r^*$ | Gurobi 3.0.0 | CPLEX 12.1 |
| 280 | 62 | 2 | 85 | 27 | | 4 | 0 | 112 | 5 | 107 | 4 | 4 | | | 107 | 1 | 1 |
| 287 | 65 | 2 | 94 | 26 | | 5 | 1 | 120 | 3 | 117 | 4 | 4 | 1 | 1 | 117 | 1 | 1 |
| 289 | 67 | 2 | 74 | 37 | | 5 | 1 | 111 | 8 | 103 | 4 | 4 | 1 | 1 | 103 | 1 | 1 |
| 304 | 71 | 3 | 104 | 43 | 3 | 7 | 2 | 150 | 3 | 147 | 5 | 5 | 73 | 895 | 147 | 9 | 2 |
| 306 | 67 | 3 | 123 | 56 | 1 | 8 | 2 | 180 | 15 | 165 | 6 | 6 | 162 | 154 | 165 | 44 | 8 |
| 311 | 70 | 3 | 102 | 52 | 9 | 7 | 2 | 163 | 31 | 132 | 5 | 5 | 49 | 392 | 132 | 15 | 2 |
| 317 | 69 | 2 | 145 | 68 | | 8 | 2 | 213 | 44 | 169 | 6 | 6 | 414 | 1598 | 169 | 64 | 15 |
| 319 | 66 | 2 | 104 | 24 | | 5 | 0 | 128 | 4 | 124 | 5 | 5 | | | 124 | 7 | 2 |
| 343 | 70 | 2 | 117 | 50 | | 6 | 1 | 167 | 20 | 147 | 5 | 5 | 13 | 4 | 147 | 15 | 4 |
| 365 | 71 | 2 | 103 | 86 | | 7 | 1 | 189 | 10 | 179 | 6 | 6 | 45 | 7 | 179 | 53 | 7 |

**Table 6.7.** Real-world results for versions $t$**-st,**$\cdot|\underline{h}$**,**$\underline{r}$**-hsh,co,sp**$|o$**-or,**$g$**-bl** – Part II

Gurobi 3.0.0 seems to outperform CPLEX 12.1. It is the other way around for the running times of the $r$-**minimizing** Binary Programm. Although both solvers determine optimal schedules with minimum number $r^*$ of railcar moves for $h^*$ humping steps within one minute, CPLEX 12.1 is even a trifle faster than Gurobi 3.0.0. In summary, for each instance an optimal schedule could be computed within 10 minutes time.

The minimum number of humping steps varies between 2 and 6, and the minimum number of railcar moves ranges from 43 to 169, depending on the complexity of the instances. On average, there are 0.41 moves – that is, 1.41 roll downs over the hump – for each railcar. Finally, let us compare the heuristically computed schedules with the optimal ones. For our instances, the worst case optimality gap is 2 humping steps and 44 railcar moves and the average gap to optimality is 0.85 humping steps and 8.9 railcar moves, see the eighth and tenth column of the Tables 6.6 and 6.7.

We discussed above results together with our partners at BASF, with the outcome that the dispatchers prefer the heuristically computed schedules, in particular, the optimal schedules of the respective **unbounded** version. What follows is some reasoning.

First of all, the advantage that the optimal schedules for the actual **bounded** case comply with the modeled fixed track lengths is in most cases of no practical use. These schedules are usually not directly applicable, since in practice the number of railcars that can be stored on a track depends on several "*soft*" parameters. For example, railcars in fact vary in length and railcars rolling down to tracks are slowed down by automatic brakes which may lead to gaps between the railcars on the track. Thus, it seems to be more reasonable to compute optimal schedules of the respective **unbounded** version and leave it to the dispatcher to handle "*full*" tracks adequately. Basically, the dispatcher has two opportunities to deal with a "*full*" track. Depending on the actual occupancy of the sorting tracks he or she may have the opportunity to redirect railcars initially planned to go on the filled track to another free sorting track in the main rail yard. Besides that, it is often possible to empty the track and park the respective railcars on a track beyond the main rail yard.

For both the **unbounded** and the exact **bounded** approach, our partner compared the quality of the computed schedules. From a practical point of view, the theoretical potential of the optimal "*bounded*" schedules for dropping the overall processing time is considered to be

small compared to the "*unbounded*" schedules. Note that for our instances the "*unbounded*" schedules realize the rearrangements with 1.44 roll downs of a railcar on average, instead of 1.41 roll downs of a railcar in case of theoretical optimality. However, considering the rule-based schedules in daily action, the "*unbounded*" schedules offer major potential for saving time.

Moreover, the computational times of less than 1 second appeal to the dispatchers. It offers the possibility of quickly reacting on any disruptions or real time changes in the order of the incoming railcars. In this sense, the "*unbounded*" approach excels by a certain *robustness*.

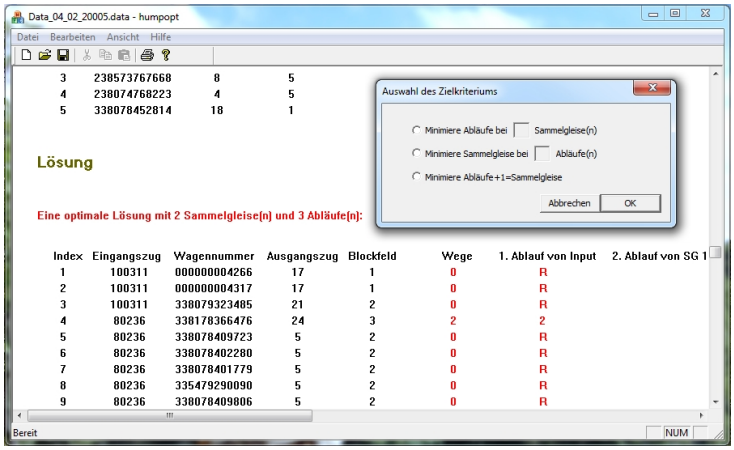


**Figure 6.1.** Snapshot of our Prototype which determines schedules for the rearrangements at BASF

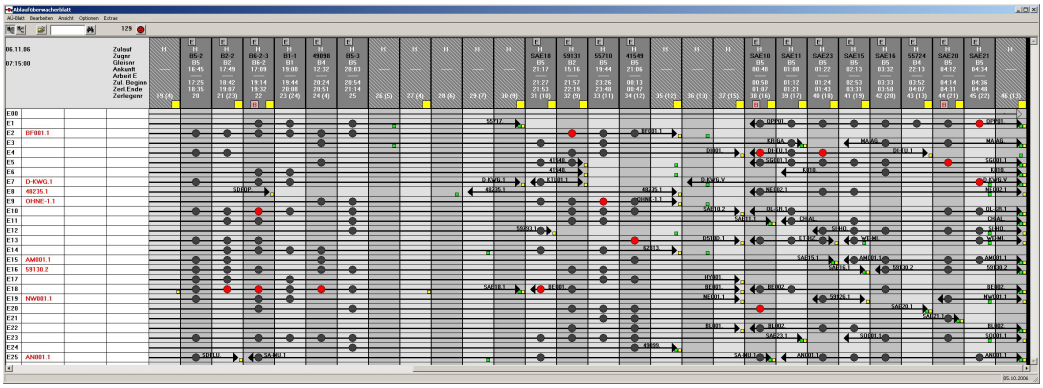

**Figure 6.2.** Snapshot of the monitoring and control system VICOS developed by Siemens

Although the "*unbounded*" schedules depend on the actual input sequence of railcars, there always is a certain structure which makes these schedules understandable. Last but not least, there is no need for commercial solvers.

We implemented and demonstrated a prototype with graphical user interface that determines schedules for the rearrangements at BASF, see Figure 6.1. Unfortunately, our approach is not yet put into daily action. The reason is, that the dispatchers cannot use it directly within the monitoring and control system VICOS, developed by Siemens and installed at BASF, see Figure 6.2. Following the proposal of our partners at BASF, we contacted Siemens Braunschweig and we together with Siemens discussed an integration of our method as add-on in VICOS. Due to several reasons, it is pending at the time being. However, we still hope that our optimization tool will be added into a major monitoring and control system like VICOS.

CHAPTER 7

# Online Versions

THIS chapter is concerned with the competitiveness of online SRS versions. The notion of competitiveness is well-known; however, the definitions in the literature are not consistent. As a consequence, we first provide some basic concepts of online optimization problems and corresponding online algorithms.

## 7.1 Definitions

For an intuition of an online problem think of the popular game Tetris. Generally, in an *online (optimization) problem* the complete instance is not known in advance but revealed gradually as a sequence of input portions. At any increase of information – that is, immediately after an input portion is given – a solution for the part of the instance known so far has to be determined. The generation of a solution for a partial instance is restricted by previously computed solutions. In most cases, prior decisions cannot be reversed at all. Each online problem implicitly involves a particular *grade of information* which is defined by the input $\mathscr{I}_0$ which is known in advance and the kind of information $\mathscr{I}_p$ that is revealed by any input portion.

An *online algorithm* is an algorithm which solves a particular online problem, that is, it outputs a series of feasible solutions for the sequence of partial instances. Of course, an online algorithm is forced to make decisions that may later turn out not to be optimal. The quality of an online algorithm can be measured by comparing the quality of the solution generated after the last input portion with the quality of the *offline optimum*. The offline optimum is an optimal solution of the

corresponding *offline problem* where the instance – that is, the entire input information – is known in advance.

**Definition 7.1**
*An online algorithm $\mathfrak{A}$ is said to be* $c$-competitive *if and only if the following inequality holds for every problem instance $\mathfrak{I}$*

$$z_{\mathfrak{A}}(\mathfrak{I}) \leq c \cdot z^*(\mathfrak{I}) + \bar{c},$$

*where $z_{\mathfrak{A}}(\mathfrak{I})$ denotes the value of the solution produced by $\mathfrak{A}$ after the last input portion for the instance $\mathfrak{I}$, $z^*(\mathfrak{I})$ is the value of the offline optimum for $\mathfrak{I}$, $c$ is any factor, and $\bar{c}$ is a constant not depending on $\mathfrak{I}$.*

We say an online algorithm is *offline-optimal* if $z_{\mathfrak{A}}(\mathfrak{I}) = z^*(\mathfrak{I})$ for any relevant instance $\mathfrak{I}$. An online algorithm $\mathfrak{A}$ is called *constant factor (c-)competitive* if $\mathfrak{A}$ is $c$-competitive for a constant $c$ not depending on $\mathfrak{I}$. The *competitive ratio of* $\mathfrak{A}$ is given by the infimum over all factors $c$ for which $\mathfrak{A}$ is $c$-competitive. The infimum over all competitive ratios of algorithms solving an online problem $\mathfrak{P}$ is called *competitive ratio of* $\mathfrak{P}$. We say $\mathfrak{P}$ as well as a $c$-competitive online algorithm for $\mathfrak{P}$ are *strongly (c-)competitive* if $c$ equals the competitive ratio of $\mathfrak{P}$. An online problem $\mathfrak{P}$ is said to be *not constant factor competitive* if its competitive ratio depends on the size of $\mathfrak{I}$, otherwise, $\mathfrak{P}$ is *constant factor competitive*. For example, an online problem $\mathfrak{P}$ whose instances consist of $n$ objects is not constant factor competitive if there is an online algorithm solving $\mathfrak{P}$ which is strongly $n/2$-competitive.

For the rest of this chapter, we deal with the online problems corresponding to the (offline) $g$**-blocks** versions of SRS. For these *online versions* we distinguish three possible grades of information. In any of these three cases, the units arrive one by one, that is, any incoming unit has immediately – before the next unit arrives – to be stored on a sorting track in the rail yard. Besides that, we always know in advance the required configuration of the output sequence. That is, in **free** or **ordered** online versions we are aware of the maximal number $g$ of blocks in the output sequence, and in $o$**-ordered** online versions we additionally know the maximal numbers $g_1, \ldots, g_o$ of blocks in the at most $o$ outbound trains. In other words, we know that each incoming unit belongs to one of $g$ groups. Note, it is possible that – after the last input portion is given – no units of a particular group arrived at all. We denote the number of groups which actually occur in the entire input of an online version by $\bar{g}$. In ($o$-)**ordered** online versions, the order

of the groups in the outbound train(s) is given by the group numbers as in the offline case. For example, if the entire input – after the last input portion – for a 2-**ordered** 5-**blocks** online version with $g_1 = 3$ and $g_2 = 2$ is given by the sequence $(5, 3, 1, 1, 4, 5)$, then, $\bar{g} = 4$ and the actual outbound trains read $(1, 1, 3)$ and $(4, 5, 5)$.

In the first of our three considered grades of information, we only know in advance the required configuration of the output sequence, that is, $\mathscr{I}_0 = \{g\}$ in **free** or **ordered** versions, or $\mathscr{I}_0 = \{g, g_1, \ldots, g_o\}$ in $o$-**ordered** versions. Besides that, for the $i$-th incoming unit we know the group it belongs to and whether or not it is the last incoming unit. In other words, the $i$-th input portion either is $\mathscr{I}_p = \{s_i, l_i\}$ for **sequence** versions or $\mathscr{I}_p = \{\mathcal{I}_i, l_i\}$ for **time windows** versions; $l_i$ is a binary which is either 1 if unit $i$ is the last one, or 0 otherwise. We say that an online version with this grade of information is an online version with sparse information.

In the second case, we additionally know if an incoming unit is the last one of a group – that is, $\mathscr{I}_p = \{s_i, l_i, l_i^{s_i}\}$ for **sequence** versions or $\mathscr{I}_p = \{\mathcal{I}_i, l_i, l_i^{s_i}\}$ for **time windows** versions – where $l_i^{s_i}$ is a binary which is either 1 if unit $i$ is the last one of group $s_i$, or 0 otherwise. For $\mathscr{I}_0$, see the first case of sparse information. A practical motivation of this grade of information is that there is a positive (maximal) demand on the number of units for any of the groups $1, \ldots, g$ in the outbound train(s). In this sense, a unit is indicated as last one, either if this unit accommodates the demand for its group – later arriving units of this group will be winnowed – or if it is for sure that no more unit of this group will arrive in the sequel. Note that we do not know in advance, if for a particular group no unit arrives at all. An online version with this grade of information is said to be an online version with little information.

In the third case, we know in advance the total number $n$ of incoming units and the numbers $\bar{n}_1, \ldots, \bar{n}_g$ of incoming units of each group. That is, $\mathscr{I}_0 = \{n, g, \bar{n}_1, \ldots, \bar{n}_g\}$ in **free** and **ordered** versions, or $\mathscr{I}_0 = \{n, g, g_1, \ldots, g_o, \bar{n}_1, \ldots, \bar{n}_g\}$ in $o$-**ordered** versions; and $\mathscr{I}_p = \{s_i\}$ or $\mathscr{I}_p = \{\mathcal{I}_i\}$, respectively. We say that an online version with this grade of information is an online version with few information.

## 7.2 Results

First of all, let us point out that there is a connection of some online $t$**-minimizing** versions of SRS to particular online graph coloring problems. In online MVC or online $b$-MES, the vertices of a graph occur one by one in an arbitrary order. With every emerging vertex we get to know its adjacency to all vertices which appeared earlier. We refer to this grade of information as online coloring information. For results on online MVC and online $b$-MES, see for example HALLDÓRSSON (1999), ERLEBACH & FIALA (2002), and LEROY-BEAULIEU (2008).

For an offline $t$**-minimizing**, $g$**-blocks** version which is equivalent to MVC ($b$-MES) of particular graphs, the corresponding online version with online coloring information is a specialization of online MVC (online $b$-MES) of these graphs where the vertices pop up in the same order as the corresponding units or groups occur in the input sequence. For example, the online versions $\underline{t}$**-st,ub**|**nsh,se,**0**-sp**|**fr,**$g$**-bl** and $\underline{t}$**-qu,ub**|**nsh,**$\cdot$**,**0**-sp**|**fr,**$g$**-bl** with little information are specializations of the online MVC of interval graphs where the intervals pop up in increasing order of their leftmost points. Note that the online algorithm FIRST FIT is offline-optimal for these online versions, whereas FIRST FIT is shown to be 8-competitive for the general online MVC of interval graphs where the intervals may pop up in any order, see NARAYANASWAMY & SUBHASH BABU (2008).

In the following we classify the competitiveness of some online versions of SRS. Let us start with a known result. The online version $\underline{t}$**-sq,ub**|**nsh,se,**$\cdot$|**or,**$n$**-bl** with few information is equivalent to online MIN COCOLORING of permutation graphs which is not constant factor competitive, see DEMANGE & LEROY-BEAULIEU (2007) and DI STEFANO ET AL. (2008). Consequently, the online versions $\underline{t}$**-sq,ub**|**nsh,se,**$\cdot$|{**or,** $o$**-or**}**,**$g$**-bl** with few information are not constant factor competitive. Now, we list a couple of observations which are easily seen. Keep in mind that the simple online algorithm assigning group $\mathfrak{g}$ to track $\mathfrak{g}$ outputs feasible solutions, and it is $g$-competitive for any online $t$**-minimizing**, **unbounded**, $g$**-blocks** version.

**Observation 7.1**
*It is well-known that* FIRST FIT *works as offline-optimal online algorithm for online MVC of permutation graphs where the vertices occur one by one according to the order of elements in the corresponding permutation, see* CHVÁTAL *(1984). This method can easily be trans-*

*lated into an offline-optimal online algorithm for the online versions* $\underline{t}$***-st,ub|nsh,se,sp|{or,*** $o$***-or},*** $g$***-bl** *and* $\underline{t}$***-qu,ub|nsh,·,sp|{or,*** $o$***-or},*** $g$***-bl** *with sparse information, due to previous results.*

**Observation 7.2**
*The online versions* $\underline{t}$***-{st,qu},ub|nsh,·,***$0$***-sp|·,*** $g$***-bl** *with sparse information are strongly* $g$*-competitive, that is, not constant factor competitive.*

**Observation 7.3**
*It is well-known that* FIRST FIT *works as offline-optimal online algorithm for online MVC of interval graphs where the intervals occur one by one and from left to right w. r. t. their leftmost points, see* CHVÁTAL *(1984). One can easily translate this method into an offline-optimal online algorithm for the online versions* $\underline{t}$***-st,ub|nsh,se,***$0$***-sp|fr,*** $g$***-bl** *and* $\underline{t}$***-qu,ub|nsh,·,***$0$***-sp|fr,*** $g$***-bl** *with little information, due to previous results.*

**Observation 7.4**
*It is easy to see that* BEST FIT *– see Algorithm 5 in Subsection 4.3.1 – works as offline-optimal online algorithm for online MVC of point-interval graphs where the triangles occur one by one and from left to right w. r. t. the corner points* $\mathfrak{p}_1^i$*. Again, this method can easily be translated into an offline-optimal online algorithm for the online versions* $\underline{t}$***-st,ub|nsh,se,***$0$***-sp|or,*** $g$***-bl** *and* $\underline{t}$***-qu,ub|nsh,·,***$0$***-sp|or,*** $g$***-bl** *with little information, due to previous results.*

**Theorem 7.1**
*The online versions* $\underline{t}$***-st,ub|nsh,co,sp|{or,*** $o$***-or},*** $g$***-bl** *with little information are strongly* $(\bar{g}-1)$*-competitive, that is, not constant factor competitive.*

*Proof.* For the online version $\underline{t}$**-st,ub|nsh,co,sp|or,**$g$**-bl**, let us consider the online algorithm $\bar{\mathfrak{A}}$ which as long as possible tries to store the first two incoming groups on track 1. At the time that becomes infeasible, $\bar{\mathfrak{A}}$ opens track 2 and spreads further units of these two groups over the tracks 1 and 2 in a feasible manner. Each of the other subsequently arriving groups is assigned to a single track $t \geq 3$ by $\bar{\mathfrak{A}}$. The number of tracks required by the schedule which is produced by $\bar{\mathfrak{A}}$ is denoted by $t_{\bar{\mathfrak{A}}}(S)$. Obviously, $t_{\bar{\mathfrak{A}}}(S) \leq \bar{g}$ for any input sequence $S$. We distinguish the two possible cases $t^*(S) = 1$ and $2 \leq t^*(S) \leq \bar{g}$ for the value $t^*(S)$ of the offline optimum. In case of $t^*(S) = 1$, it follows with $t_{\bar{\mathfrak{A}}}(S) = \bar{g} - 1$ that $t_{\bar{\mathfrak{A}}}(S) \leq (\bar{g}-1) \cdot t^*(S)$. In case of $2 \leq t^*(S) \leq \bar{g}$, we

get $t_{\bar{\mathfrak{A}}}(S) \leq \bar{g} = 2\bar{g} - \bar{g} \leq t^*(S) \cdot \bar{g} - t^*(S) = (\bar{g} - 1) \cdot t^*(S)$. As a consequence, online algorithm $\bar{\mathfrak{A}}$ is $\bar{g} - 1$-competitive for the online version $\underline{t}$**-st,ub**|**nsh,co,sp**|**or,**$g$**-bl**.

The first $\bar{g} - 1$ incoming units of an input sequence $(2, 3, \ldots, \bar{g}, ?)$ need to be stored on $\bar{g} - 1$ different tracks by any online algorithm which ensures feasible solutions for subsequently arriving units. If we occupy less than $\bar{g} - 1$ tracks, then the input $(2, 3, \ldots, \bar{g}, 1)$ would lead to infeasibility. However, for the input $(2, 3, \ldots, \bar{g}, \bar{g} + 1)$ the offline optimum corresponds to storing all units on a single track. As a consequence, the competitive ratio of the online version $\underline{t}$**-st,ub**|**nsh,co,sp**|**or,**$g$**-bl** with little information is $\bar{g} - 1$. The result for the respective $o$**-ordered** version is analogously shown. □

**Theorem 7.2**
*The online versions* $\underline{t}$***-st,ub*|*nsh,*{*co,tw*}*,*0***-sp***|·,*g*-bl** *with few information are strongly* $g$*-competitive, that is, not constant factor competitive.*

*Proof.* For the online version $\underline{t}$**-st,ub**|**nsh,co,**0**-sp**|**fr,**$g$**-bl** as well as the online version $\underline{t}$**-st,ub**|**nsh,co,**0**-sp**|**or,**$g$**-bl**, the first $g$ incoming units of an input sequence $(g, g - 1, \ldots, 1, ?)$ need to be stored on $g$ different tracks to ensure feasible solutions for further incoming units. Assume that we occupy less than $g$ tracks. Then, the input $(g, g - 1, \ldots, 1, g, g - 1, \ldots, 1)$ would lead to infeasibility in case of $\underline{t}$**-st,ub**|**nsh,co,**0**-sp**|**fr,**$g$**-bl** and $(g, g - 1, \ldots, 1, 2, 3, \ldots, g, 1)$ would do for $\underline{t}$**-st,ub**|**nsh,co,**0**-sp**|**or,**$g$**-bl**. However, for the input sequence $(g, g - 1, \ldots, 1, 1, 2, \ldots, g)$, the offline optimum corresponds to storing all units on a single track. As a consequence, the competitive ratio of the online versions $\underline{t}$**-st,ub**|**nsh,co,**0**-sp**|**fr,**$g$**-bl** and $\underline{t}$**-st,ub**|**nsh,co,**0**-sp**|{**or,** $o$**-or**}**,**$g$**-bl** with few information is $g$.

For the online version $\underline{t}$**-st,ub**|**nsh,tw,**0**-sp**|**or,**$g$**-bl** the first $g$ known intervals $[1, 4g - 2 \cdot 1], [2, 4g - 2 \cdot 2], [3, 4g - 2 \cdot 3], \ldots, [g, 4g - 2 \cdot g]$ of incoming units require $g$ different tracks to ensure feasibility. Again, if less than $g$ tracks are occupied, then, the complete input $[1, 4g - 2 \cdot 1]$, $[2, 4g - 2 \cdot 2]$, $[3, 4g - 2 \cdot 3]$, $\ldots$, $[g, 4g - 2 \cdot g]$, $[g + 1, 4g - 2 \cdot (g - 1)]$, $[g + 2, 4g - 2 \cdot (g - 2)]$, $\ldots$, $[2g - 1, 4g - 2 \cdot 1]$ would cause infeasibility. However, if the complete input reads $[1, 4g - 2 \cdot 1]$, $[2, 4g - 2 \cdot 2]$, $[3, 4g - 2 \cdot 3]$, $\ldots$, $[g, 4g - 2 \cdot g]$, $[2g + 1, 4g - 2 \cdot (g - 1)]$, $[2(g + 1) + 1, 4g - 2 \cdot (g - 2)]$, $\ldots$, $[2(2g - 2) + 1, 4g - 2 \cdot 1]$, then the offline optimum corresponds to storing all units on a single track. Thus, the competitive ratio of the online version $\underline{t}$**-st,ub**|**nsh,tw,**0**-sp**|**or,**$g$**-bl** with few information is $g$. □

Of course, the previously shown competitive ratios of the **unbounded** online versions provide lower bounds on the competitive ratios of the corresponding $b$-**bounded** online versions for general $b$. Consequently,

| | | |
|---|---|---|
| $\underline{t}$-{**st,qu**},$b$-**bd**\|**nsh,**$\cdot$,0-**sp**\|$\cdot$,$g$-**bl** | with | sparse information, |
| $\underline{t}$-**st,**$b$-**bd**\|**nsh,co,sp**\|{**or,** $o$-**or**},$g$-**bl** | with | little information, |
| $\underline{t}$-**st,**$b$-**bd**\|**nsh,**{**co,tw**},0-**sp**\|$\cdot$,$g$-**bl** | with | few information, |
| $\underline{t}$-**sq,**$b$-**bd**\|**nsh,se,**$\cdot$\|{**or,** $o$-**or**},$g$-**bl** | with | few information |

are not constant factor competitive for general $b$.

On the positive side for the $b$-**bounded** case, there is the following result. The online versions $\underline{t}$-**st,**$b$-**bd**\|**nsh,se,sp**\|{**or,** $o$-**or**},$g$-**bl** and $\underline{t}$-**qu,**$b$-**bd**\|**nsh,**$\cdot$**,sp**\|{**or,** $o$-**or**},$g$-**bl** with sparse information are strongly $2 - 1/l$-competitive, where $l = \min\{b, t^*(S)\}$ and $t^*(S)$ is the value of the offline optimum. This is due to a result from DEMANGE ET AL. (2008), which says that FIRST FIT works as a strongly $2 - 1/l$-competitive online algorithm for the online $b$-MES of permutation graphs where the vertices occur one by one according to the order of the elements in the corresponding permutation.

Finally, let us deal with the $h$-**hump-shunting** online versions $t$-**st,**$\cdot$\|$\underline{h}$-**hsh,co,sp**\|{**or,** $o$-**or**},$g$-**bl** that relate to the real-world optimization problem at BASF. Note, as in the offline case we do not allow any humping steps before the last incoming unit – given by the last input portion – is assigned to a track. Besides that, we assume that there are arbitrarily many sorting tracks available in the rail yard.

For the online versions $t$-**st,**$b$-**bd**\|$\underline{h}$-**hsh,co,sp**\|{**or,** $o$-**or**},$g$-**bl** with fixed $b$, the online algorithm $\tilde{\mathfrak{A}}$ proceeds as follows. It assigns an incoming unit directly to an output track whenever this ensures feasibility. We denote the number of units which are directly moved to the output track(s) by $\hat{n}_1$. With all other incoming units $\tilde{\mathfrak{A}}$ iteratively fills the tracks $1, \ldots, \left\lceil \frac{n-\hat{n}_1}{b} \right\rceil =: \tilde{t}$, that is, first track 1 is filled, after that track 2, and so on. The set of units parked on track $i$ is given by the subsequence $S_i$ of the input sequence $S$.

The solution produced by $\tilde{\mathfrak{A}}$ performs the first $\tilde{t}$ humping steps according to the track execution order $E = (1, \ldots, \tilde{t}, ?)$. During these humping steps the units roll down in the same order as they occur in the input sequence, since $\bar{S} = (S_1 \oplus S_2 \oplus \cdots \oplus S_{\tilde{t}})$ is a subsequence of $S$. Before the first humping step is actually executed, $\tilde{\mathfrak{A}}$ determines the

offline optimum $O_{\bar{S}}$ – regarding the considered version – for $\bar{S}$. Note that the value $h^*(\bar{S})$ of this solution is not greater than the value $h^*(S)$ of the offline optimum for $S$. As above-mentioned, the solution of $\tilde{\mathfrak{A}}$ performs the first $\tilde{t}$ humping steps from the tracks $1,\ldots,\tilde{t}$ according to $O_{\bar{S}}$ such that the units only roll on the sorting tracks $\tilde{t}+1,\ldots,\tilde{t}+h^*(\bar{S})$, or on the output track(s). After that, we proceed with the humping steps – described by $O_{\bar{S}}$ – from the tracks $\tilde{t}+1,\ldots,\tilde{t}+h^*(\bar{S})$.

After all, $\tilde{\mathfrak{A}}$ determines a feasible solution with

$$h_{\tilde{\mathfrak{A}}}(S) = \left\lceil \frac{n-\hat{n}_1}{b} \right\rceil + h^*(\bar{S}) \leq \left\lceil \frac{n-\hat{n}_1}{b} \right\rceil + h^*(S)$$

humping steps according to the track execution order $E = (1,\ldots,\tilde{t}, \tilde{t}+1,\ldots,\tilde{t}+h^*(\bar{S}))$.

The **unbounded** online versions $t$**-st,ub**|$\underline{h}$**-hsh,co,sp**|{**or,** $o$**-or**}**,**$g$**-bl** are equivalent to the corresponding $b$**-bounded** versions with a fixed track length $b \geq n$. As a consequence, we – due to $\left\lceil \frac{n-\hat{n}_1}{b} \right\rceil \leq 1$ – observe what follows for the **unbounded** case.

**Observation 7.5**
*The online versions $t$**-st,ub**|$\underline{h}$**-hsh,co,sp**|{**or,** $o$**-or**}**,**$g$**-bl** are strongly 1-competitive for any grade of information. In particular, we need at most one humping step more than the offline optimum.*

Note that there is no offline-optimal online algorithm for the online versions $t$**-st,ub**|$\underline{h}$**-hsh,co,sp**|{**or,** $o$**-or**}**,**$g$**-bl** with little information. For example, consider the online versions $t$**-st,ub**|$\underline{h}$**-hsh,co,sp**|**or,**3**-bl**. Assume that the first incoming unit is of group 2. We need to store it on a sorting track because we do not know whether a unit of group 1 will arrive afterwards. Thus, we have to perform at least one humping step. However, if the entire input sequence reads $(2,3)$, then no humping step is necessary at all.

Let us go back to the $b$**-bounded** case. In the following we restrict these online versions to input sequences $S$ with $h^*(S) \geq 1$. In other words, at least one humping step is needed for the required rearrangement of the input sequence in the offline case.

**Theorem 7.3**
*The online versions $t$**-st,**$b$**-bd**|$\underline{h}$**-hsh,co,sp**|{**or,** $o$**-or**}**,**$g$**-bl** with little information are strongly $\lceil n/b \rceil$-competitive – that is, not constant factor competitive – for any fixed $b$.*

*Proof.* As described above, the online algorithm $\tilde{\mathfrak{A}}$ determines a feasible solution with $h_{\tilde{\mathfrak{A}}}(S) \leq \lceil n/b \rceil + h^*(S)$ humping steps.
In case of $\lceil \frac{n}{b} \rceil = 1$ (**unbounded**), we get $h_{\tilde{\mathfrak{A}}}(S) \leq \lceil \frac{n}{b} \rceil \cdot h^*(S) + 1$.
In case of $\lceil \frac{n}{b} \rceil \geq 2$, $h^*(S) = 1$, we get $h_{\tilde{\mathfrak{A}}}(S) \leq \lceil \frac{n}{b} \rceil \cdot h^*(S) + 1$.
In case of $\lceil \frac{n}{b} \rceil \geq 2$, $h^*(S) \geq 2$, we get $h_{\tilde{\mathfrak{A}}}(S) \leq \lceil \frac{n}{b} \rceil \cdot h^*(S)$.
Thus, online algorithm $\tilde{\mathfrak{A}}$ is $\lceil n/b \rceil$-competitive.
For the online version $t$**-st,**$b$**-bd**$|\underline{h}$**-hsh,co,sp**$|$**or,**4**-bl** with little information, any online algorithm has to store the entire input sequence $(2, 2, \ldots, 2, 4, 3)$ on the sorting tracks, because until the last input portion is revealed it is not known whether a unit of group 1 will arrive. Thus, the solution obtained by any online algorithm requires at least $\lceil n/b \rceil$ humping steps. The offline optimum needs only one humping step. Hence, the online versions $t$**-st,**$b$**-bd**$|\underline{h}$**-hsh,co,sp**$|\{$**or,** $o$**-or**$\}$**,**$g$**-bl** with little information are strongly $\lceil n/b \rceil$-competitive. □

In the online versions $t$**-st,**$b$**-bd**$|\underline{h}$**-hsh,co,sp**$|\{$**or,** $o$**-or**$\}$**,**$g$**-bl** with few information, it is possible to move all the units from the input track to the output track(s) which are directly assigned to the output track(s) in the offline optimum. In other words, online algorithm $\tilde{\mathfrak{A}}$ is implementable such that the number $\hat{n}_1$ is best possible. Consequently, it is $\lceil (n - \hat{n}_1)/b \rceil \leq h^*(S)$, which implies the following result.

**Theorem 7.4**
*The online algorithm $\tilde{\mathfrak{A}}$ is 2-competitive for the two online versions* $t$***-st,**$b$**-bd**$|\underline{h}$**-hsh,co,sp**$|\{$**or,** $o$**-or**$\}$**,**$g$**-bl** *with few information.*

From practical point of view, the results presented in this chapter can be summarized as follows. For most versions of SRS, it is in general not avoidable that the quality of the rearrangements performed in the online case with sparse information – the knowledge of the incoming sequence reveals gradually by the units arriving one by one – is arbitrarily bad compared to the best possible solution (of the offline case). This is also true for the online version of our real-world optimization problem at BASF, even in case of little information.

CHAPTER 8

# Conclusion

THIS thesis is concerned with the problem of optimally rearranging objects, in particular, railcars in a rail yard. We introduced a thorough classification for versions of such rearrangement problems. For various versions,

- we provided a translation into a mathematical language,
- we listed known results,
- we showed their equivalence to certain graph coloring, scheduling, and bin packing problems,
- we classified their computational complexity,
- we gave approximability results,
- and we proposed solution approaches for determining good or even optimal schedules.

Computational results show that we can determine optimal schedules for real input data with surprisingly small computational effort. This is true for all considered versions; in particular, for the version that corresponds to the real-world optimization problem dealt with in our project together with BASF.

---

Let us conclude with some recommendations for future work. In this thesis, we investigated many – however, by far not all – applicable offline versions of SRS. Besides that, our competitive analysis of online

versions is of an introductory nature. That is, quite a lot of offline and online SRS versions still offer challenging open questions for future research.

Ultimately, the two open theoretical questions related to this thesis which are of the greatest interest to me read as follows.

- What is the computational complexity of the three equivalent versions $\underline{t}$**-st,ub**|**nsh,se,sp**|**fr,**$g$**-bl**, $\underline{t}$**-qu,ub**|**nsh,se,sp**|**fr,**$g$**-bl**, and $\underline{t}$**-qu,ub**|**nsh,co,sp**|**fr,**$g$**-bl**?
- Is MVC of polygon-circle graphs constant factor approximable – or even 2-approximable – in polynomial time?

# Bibliography


ACHTERBERG, T. (2007). *Constraint Integer Programming*. Ph.D. thesis, Technische Universität Berlin.

AGEEV, A. A. (1996). A triangle-free circle graph with chromatic number 5. *Discrete Mathematics*, *152*(1-3), 295–298.

ALBERT, M. H., ATKINSON, M. D., & LINTON, S. A. (2010). Permutations generated by stacks and deques. *Annals of Combinatorics*, *14*(1), 3–16.

ALDRED, R. E. L., ATKINSON, M. D., & MCCAUGHAN, D. J. (2010). Avoiding consecutive patterns in permutations. *Advances in Applied Mathematics*, *45*(3), 449–461.

ASSAD, A. A. (1981). Analytical models in rail transportation: An annotated bibliography. *INFOR*, *19*(1), 59–80.

ASSAD, A. A. (1983). Analysis of rail classification policies. *INFOR*, *21*(4), 293–314.

AUSIELLO, G., CRESCENZI, P., GAMBOSI, G., KANN, V., MARCHETTI-SPACCAMELA, A., & PROTASI, M. (1999). *Complexity and Approximation*. Springer, Berlin.

AVRIEL, M., PENN, M., & SHPIRER, N. (2000). Container ship stowage problem: Complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics*, *103*(1), 271–179.

BAR-YEHUDA, R., & FOGEL, S. (1998). Partitioning a sequence into few monotone subsequences. *Acta Informatica*, *35*(5), 421–440.

BAUMANN, O. (1959). Die Planung der Simultanformation von Nahgüterzügen für den Rangierbahnhof Zürich-Limmattal. *Rangiertechnik*, *Nr. 19*, 25–35.

BERGE, C. (1961). Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, *10*, 114–115.

BODLAENDER, H. L., & JANSEN, K. (1995). Restrictions of graph partition problems. Part I. *Theoretical Computer Science*, *148*(1), 93–109.

BONA, M. (2004). *Combinatorics of Permutations*. CRC Press, Inc., Boca Raton, FL.

BORNDÖRFER, R., & CARDONHA, C. (2009). A set partitioning approach to shunting. *Electronic Notes in Discrete Mathematics*, *35*, 359–364.

BRANDSTÄDT, A., & KRATSCH, D. (1986). On partitions of permutations into increasing and decreasing subsequences. *Elektronische Informationsverarbeitung und Kybernetik*, *22*(5/6), 263–273.

BRANDSTÄDT, A., LE, V. B., & SPINRAD, J. P. (1999). *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

CAMPÊLO, M., CAMPOS, V. A., & CORRÊA, R. (2008). On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, *156*(7), 1097–1111.

CAMPÊLO, M., CORRÊA, R., & FROTA, Y. (2004). Cliques, holes and the vertex coloring polytope. *Information Processing Letters*, *89*(4), 159–164.

CARAMIA, M., & DELL'OLMO, P. (2001). Iterative coloring extension of a maximum clique. *Naval Research Logistic*, *48*(6), 518–550.

CARAMIA, M., & DELL'OLMO, P. (2004). Bounding vertex coloring by truncated multistage branch and bound. *Networks*, *44*(4), 231–242.

CESELLI, A., GATTO, M. J., LÜBBECKE, M. E., NUNKESSER, M., & SCHILLING, H. (2008). Optimizing the cargo express service of swiss federal railways. *Transportation Science*, *42*(4), 450–465.

CHUNG, F. R. K. (1980). On unimodal subsequences. *Journal of Combinatorial Theory, Series A*, *29*(3), 267–279.

CHVÁTAL, V. (1984). Perfectly ordered graphs. In C. Berge, & V. Chátal (Eds.) *Topics on Perfect Graphs*, Annals of Discrete Mathematics, 21, (pp. 63–65). North-Holland, Amsterdam.

COLL, P., MARENCO, J., MENDEZ-DIAZ, I., & ZABALA, P. (2002). Facets of the graph coloring polytope. *Annals of Operations Research*, *116*(12), 79–90.

CORNEIL, D. G., & KAMULA, P. A. (1987). Extensions of permutation and interval graphs. In *Combinatorics, Graph Theory, and Computing, Proceedings of the 18th Southeastern Conference on Combinatorics, Boca Raton, FL*, Congressus Numerantium 58, (pp. 267–275). Utilitas Mathematica, Winnipeg.

CORNELSEN, S., & DI STEFANO, G. (2004). Platform assignment. In A. Schöbel, & F. H. Wagner (Eds.) *Proceedings of the 4th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2004)*, Lecture Notes in Computer Science, (pp. 233–245). Springer, Berlin.

CORNELSEN, S., & DI STEFANO, G. (2007). Track assignment. *Journal of Discrete Algorithms*, *5*(2), 250–261.

DAGAN, I., GOLUMBIC, M. C., & PINTER, R. Y. (1988). Trapezoid graphs and their coloring. *Discrete Applied Mathematics*, *21*(1), 35–46.

DAGANZO, C. F. (1986). Static blocking at railyards: Sorting implications and track requirements. *Transportation Science*, *20*(3), 189–199.

DAGANZO, C. F. (1987A). Static blocking at railyards: Part I. Heterogeneous traffic. *Transportation Research*, *Part B 21*(1), 29–40.

DAGANZO, C. F. (1987B). Static blocking at railyards: Part I. Homogeneous traffic. *Transportation Research*, *Part B 21*(1), 1–27.

DAGANZO, C. F., DOWLING, R. G., & HALL, R. W. (1983). Railroad classification yard throughput: The case of multistage triangular sorting. *Transportation Research*, *Part A 17*(2), 95–106.

DAHLHAUS, E., HORAK, P., MILLER, M., & RYAN, J. F. (2000A). The train marshalling problem. *Discrete Applied Mathematics*, *103*(1–3), 41–54.

DAHLHAUS, E., MANNE, F., MILLER, M., & RYAN, J. F. (2000B). Algorithms for combinatorial problems related to train marshalling. In *Proceedings of AWOCA 2000*, (pp. 7–16). Hunter Valley, NSW.

DEMANGE, M., DI STEFANO, G., & LEROY-BEAULIEU, B. (2008). Online bounded coloring of permutation and overlap graphs. *Electronic Notes in Discrete Mathematics*, *30*, 213–218.

DEMANGE, M., EKIM, T., & DE WERRA, D. (2009). A tutorial on the use of graph coloring for some problems in robotics. *European Journal of Operational Research*, *192*(1), 41–55.

DEMANGE, M., & LEROY-BEAULIEU, B. (2007). Online coloring of comparability graphs: Some results, ORWP 07/01, EPFL Lausanne.

DI STEFANO, G., & KOČI, M. L. (2004). A graph theoretical approach to the shunting problem. *Electronic Notes in Theoretical Computer Science*, *92*, 16–33.

DI STEFANO, G., KRAUSE, S., LÜBBECKE, M. E., & ZIMMERMANN, U. T. (2006). On minimum $k$-modal partitions of permutations. In J. R. Correa, A. Hevia, & M. Kiwi (Eds.) *Latin American Theoretical Informatics (LATIN2006)*, vol. 3887 of *Lecture Notes in Computer Science*, (pp. 374–385). Springer, Berlin.

DI STEFANO, G., KRAUSE, S., LÜBBECKE, M. E., & ZIMMERMANN, U. T. (2008). On minimum k-modal partitions of permutations. *Journal of Discrete Algorithms*, *6*(3), 381–392.

EGGERMONT, C., HURKENS, C. A. J., MODELSKI, M., & WOEGINGER, G. J. (2009). The hardness of train rearrangements. *Operations Research Letters*, *37*(2), 80–82.

EPSTEIN, L., & LEVIN, A. (2008). On bin packing with conflicts. *Journal on Optimization*, *19*(3), 1270–1298.

ERDŐS, P., & SZEKERES, G. (1935). A combinatorial problem in geometry. *Compositio Mathematica*, *2*, 463–470.

ERLEBACH, T., & FIALA, J. (2002). On-line coloring of geometric intersection graphs. *Computational Geometry: Theory and Applications*, *23*(2), 243–255.

EVEN, S., & ITAI, A. (1971). Queues, stacks and graphs. In Z. Kohavi, & A. Paz (Eds.) *Theory of Machines and Computations*, (pp. 71–86). Academic Press, New York, NY.

FELSNER, S., MÜLLER, R., & WERNISCH, L. (1997). Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, *74*(1), 13–32.

FERTIG, H. R. (1927). System classification plan improves yard efficiency. *Railway Age*, *19. February*, 515–522.

FLANDORFFER, H. (1953). Vereinfachte Güterzugbildung. *Rangiertechnik*, *Nr. 13*, 114–118.

FOMIN, F. V., KRATSCH, D., & NOVELLI, J.-C. (2002). Approximating minimum cocolourings. *Information Processing Letters*, *84*(5), 285–290.

FRANK, A. (1976). Some polynomial algorithms for certain graphs and hypergraphs. In C. Nash-Williams, & J. Sheehan (Eds.) *Proceedings of the Fifth British Combinatorial Conference, 1975*, (pp. 211–226). Utilitas Mathematica, Winnipeg.

FRELING, T., LENTINK, R. M., KROON, L. G., & HUISMAN, D. (2005). Shunting of passenger train units in a railway station. *Transportation Science*, *39*(2), 261–272.

FULKERSON, D. R., & GROSS, O. A. (1965). Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, *15*(3), 835–855.

GALLO, G., & DI MIELE, F. (2001). Dispatching buses in parking depots. *Transportation Science*, *35*(3), 322–330.

GAREY, M. R., & JOHNSON, D. S. (1979). *Computers and intractability*. W. H. Freeman and Co., San Francisco, CA.

GAREY, M. R., JOHNSON, D. S., MILLER, G. L., & PAPADIMITRIOU, C. H. (1980). The complexity of coloring circular arcs and chords. *Journal on Algebraic and Discrete Methods*, *1*, 216–227.

GATTO, M., MAUE, J., MIHALÁK, M., & WIDMAYER, P. (2009). Shunting for dummies: An introductory algorithmic survey. In R. Ahuja, R. Möhring, & C. Zaroliagis (Eds.) *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, (pp. 310–337). Springer, Berlin Heidelberg.

GAVRIL, F. (1973). Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks*, *3*, 261–273.

GAVRIL, F. (2000). Maximum weight independent sets and cliques in intersection graphs of filaments. *Information Processing Letters*, *73*(5-6), 181–188.

GOLUMBIC, M. C. (2004). *Algorithmic Graph Theory and Perfect Graphs*. North-Holland, Amsterdam.

GRASSMANN, E. (1952). Zur Geschichte der Rangiertechnik. *Rangiertechnik*, *Nr. 12*, 9–18.

GROSS, J. L., & YELLEN, J. (2005). *Graph Theory and Its Applications, Second Edition*. Chapman & Hall/CRC, London.

HAJÖS, G. (1957). Über eine Art von Graphen. *Internationale Mathematische Nachrichten*, *11*, 65.

HALLDÓRSSON, M. M. (1999). Online coloring known graphs. In *SODA '99: Proceedings of the tenth annual ACM-SIAM Symposium on Discrete Algorithms*, (pp. 917–918). Society for Industrial and Applied Mathematics, Philadelphia, PA.

HAMDOUNI, M., DESAULNIERS, G., & SOUMIS, F. (2007). Parking buses in a depot using block patterns: A Benders decomposition approach for minimizing type mismatches. *Computers & Operations Research*, *34*(11), 3362–3379.

HAN, Y.-H. (2004). *Dynamic Sequencing of Jobs on Conveyor Systems for Minimizing Changeovers*. Ph.D. thesis, Georgia Institute of Technology, School of Industrial and Systems Engineering.

HANSEN, P., LABBÉ, M., & SCHINDL, D. (2009). Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results. *Discrete Optimization*, *6*(2), 135–147.

HANSMANN, R. S., & ZIMMERMANN, U. T. (2008). Optimal sorting of rolling stock at hump yards. In W. Jäger, & H.-J. Krebs (Eds.) *Mathematics–Key Technology for the Future: Joint Projects Between Universities and Industry*, (pp. 189–203). Springer, Berlin.

HAUSER, A., & MAUE, J. (2010). Experimental evaluation of approximation and heuristic algorithms for sorting railway cars. In P. Festa

(Ed.) *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA-10)*, vol. 6049 of *LNCS*, (pp. 154–165). Springer, Berlin.

HIRSCHBERG, D. S., & RÉGNIER, M. (2000). Tight bounds on the number of string subsequences. *Journal of Discrete Algorithms*, *1*(1), 123–132.

IVIĆ, M., MARKOVIĆ, M., & MARKOVIĆ, A. (2007). Effects of the application of conventional methods in the process of forming the pick-up trains. *Yugoslav Journal of Operations Research*, *17*(2), 245–256.

JACOB, R. (2007). On shunting over a hump. Unpublished.

JACOB, R., MÁRTON, P., MAUE, J., & NUNKESSER, M. (2007). Multistage methods for freight train classification. In C. Liebchen, R. K. Ahuja, & J. A. Mesa (Eds.) *ATMOS 2007 - 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, (pp. 158–174). IBFI Schloss Dagstuhl, Wadern.

JACOB, R., MÁRTON, P., MAUE, J., & NUNKESSER, M. (2011). Multistage methods for freight train classification. In *NETWORKS–Special Issue: Optimization in Scheduled Transportation Networks*, vol. 57 (1), (pp. 87–105). Wiley, New York, NY.

JANSEN, K. (2003). The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computation*, *180*(2), 71–81.

JOHNSON, D. S. (1973). Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the fifth annual ACM Symposium on Theory of Computing*, (pp. 38–49). Association for Computing Machinery, New York, NY.

KAGARIS, D., & TRAGOUDAS, S. (1999). Maximum weighted independent sets on transitive graphs and applications. *Integration, the VLSI Journal*, *27*(1), 77–86.

KNUTH, D. E. (1973). *Fundamental Algorithms, The Art of Computer Programming. Vol. 1 (Second Edition)*. Addison-Wesley, Reading, MA.

KOEBE, M. (1992). On a new class of intersection graphs. In M. Fiedler, & J. Nesetril (Eds.) *Graphs and Complexity, Proceedings*

*of the 4th Czechoslovak Symposium on Combinatorics, Prachatice, 1992, Annals of Discrete Mathematics*, vol. 51, (pp. 141–143). North-Holland, Amsterdam.

KÖNIG, F. G. (2009). *Sorting with Objectives - Graph Theoretic Concepts in Industrial Optimization*. Ph.D. thesis, Technische Universität Berlin.

KÖNIG, F. G., & LÜBBECKE, M. E. (2008). Sorting with complete networks of stacks. In S.-H. Hong, H. Nagamochi, & T. Fukunaga (Eds.) *International Symposium on Algorithms and Computation (ISAAC 2008), Lecture Notes in Computer Science 5369*, (pp. 896–907). Springer, Berlin.

KÖNIG, F. G., LÜBBECKE, M. E., MÖHRING, R. H., SCHÄFER, G., & SPENKE, I. (2007). Solutions to real-world instances of PSPACE-complete stacking. In L. Arge, M. Hoffmann, & E. Welzl (Eds.) *Proceedings of the 15th European Symposium on Algorithms (ESA)*, Lecture Notes in Computer Science, (pp. 729–740). Springer, Berlin.

KÖNIG, H., & SCHALTEGGER, P. (1967). Optimale Simultanformation von Nahgüterzügen in Rangierbahnhöfen. *Monatsschrift der Internationalen Eisenbahn-Kongress-Vereinigung*, *4*(1), 1–18.

KORTE, B., & VYGEN, J. (2008). *Combinatorial Optimization*. Springer, Berlin.

KOSTOCHKA, A., & KRATOCHVIL, J. (1997). Covering and coloring polygon-circle graphs. *Discrete Mathematics*, *163*, 299–305.

KRELL, K. (1962). Grundgedanken des Simultanverfahrens. *Rangiertechnik*, *Nr. 22*, 15–23.

KRELL, K. (1963). Ein Beitrag zur gemeinsamen Bildung von Nahgüterzügen. *Rangiertechnik*, *Nr. 23*, 16–25.

KROON, L. G., LENTINK, R. M., & SCHRIJVER, A. (2006). Shunting of passenger train units: An integrated approach. Tech. rep., ARRIVAL Project.

LEROY-BEAULIEU, B. (2008). *Some Coloring and Walking Problems in Graphs*. Ph.D. thesis, Ecole Polytechnique Federale de Lausanne, Switzerland.

LÜBBECKE, M. E., & ZIMMERMANN, U. T. (2000). Optimale Disposition von Rangierlokomotiven bei Werks- und Industriebahnen – vom Bedarf zum Produkt. In *Arbeitskreis Mathematik in Forschung und Praxis: 18. Symposium "Modellierung und Simulation von Verkehr", Bad Honnef am 17./18.11.1999*, (pp. 35–40).

LÜBBECKE, M. E., & ZIMMERMANN, U. T. (2005). Shunting minimal rail car allocation. *Computational Optimization and Applications*, *31*(3), 295–308.

LUCET, C., MENDES, F., & MOUKRIM, A. (2006). An exact method for graph coloring. *Computers and Operations Research*, *33*(8), 2189–2207.

MARCZEWSKI, E. (1945). Sur deux propriétés des classes d'ensembles. *Fundamenta Mathematicae*, *33*, 303–307.

MÁRTON, P., MAUE, J., & NUNKESSER, M. (2009). An improved classification procedure for the hump yard Lausanne Triage. In J. Clausen, & G. Di Stefano (Eds.) *Proceedings of the 9-th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS-09)*, (pp. 1–15). IBFI Schloss Dagstuhl, Wadern.

MEHROTRA, A., & TRICK, M. A. (1996). A column generation approach for graph coloring. *INFORMS Journal on Computing*, *8*(4), 344–354.

MÉNDEZ-DÍAZ, I., & ZABALA, P. (2001). A polyhedral approach for graph coloring. *Electronic Notes in Discrete Mathematics*, *7*, 1–4.

MÉNDEZ-DÍAZ, I., & ZABALA, P. (2006). A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, *154*(5), 826–847.

MÉNDEZ-DÍAZ, I., & ZABALA, P. (2008). A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, *156*(2), 159–179.

NARAYANASWAMY, N. S., & SUBHASH BABU, R. (2008). A note on First-Fit coloring of interval graphs. *Order*, *25*(1), 49–53.

NEMHAUSER, G. L., & WOLSEY, L. A. (1988). *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY.

PALUBECKIS, G. (2008). On the graph coloring polytope. *Information Technology And Control*, *37*(1), 7–11.

PAPADIMITRIOU, C. H. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA.

PAPADIMITRIOU, C. H., & STEIGLITZ, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover, Mineola, NY.

PENTINGA, K. J. (1959). Teaching simultaneous marshalling. *The Railway Gazette*, *110*(21), 590–593.

PETERSEN, E. R. (1977A). Railyard modeling: Part I. Prediction of put-through time. *Transportation Science*, *11*(1), 37–49.

PETERSEN, E. R. (1977B). Railyard modeling: Part II. The effect of yard facilities. *Transportation Science*, *11*(1), 50–59.

PNUELI, A., LEMPEL, A., & EVEN, S. (1971). Transitive orientation of graphs and identification of permutation graphs. *Canadian Journal of Mathematics*, *23*, 160–175.

PRATT, V. R. (1973). Computing permutations with double-ended queues, parallel stacks and parallel queues. In *STOC '73: Proceedings of the 5th annual ACM Symposium on Theory of Computing*, (pp. 268–277). Association for Computing Machinery, New York, NY.

SCHALTEGGER, P. (1967). Optimierung eines Rangierverfahrens. *Ablauf- und Planungsforschung*, *8*(4), 302–314.

SCHRIJVER, A. (2003). *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin.

SEWELL, E. C. (1993). An improved algorithm for exact graph coloring. In D. S. Johnson, & M. A. Trick (Eds.) *Cliques, Coloring, and Satisfiability: Proceedings of the Second DIMACS Implementation Challenge*, (pp. 359–373). American Mathematical Society, Boston, MA.

SIDDIQEE, M. W. (1972). Investigation of sorting and train formation schemes for a railroad hump yard. In G. F. Newell (Ed.) *Proceedings of the 5th International Symposium on the Theory of Traffic Flow and Transportation*, (pp. 377–388). American Elsevier, New York, NY.

SMITH, R., & VATTER, V. (2009). The enumeration of permutations sortable by pop stacks in parallel. *Information Processing Letters*, *109*(12), 626–629.

SPIERLING, C. (2007). *Optimales Sortieren von Güterwaggons in Gruppen*. Diploma thesis, Institut für Mathematische Optimierung, Technische Universität Carolo-Wilhelmina zu Braunschweig.

STEELE, J. M. (1995). Variations on the monotone subsequence theme of Erdős and Szekeres. In D. Aldous, P. Diaconis, J. Spencer, & J. M. Steele (Eds.) *Discrete Probability and Algorithms*, (pp. 111–131). Springer, New-York.

TARJAN, R. E. (1972). Sorting using networks of queues and stacks. *Journal of the Association for Computing Machinery*, *19*(2), 341–346.

TARJAN, R. E., & YANNAKAKIS, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, *13*(3), 566–579.

UNGER, W. (1988). On the $k$-colouring of circle graphs. In R. Cori, & M. Wirsing (Eds.) *STACS 88: Proceedings of the 5th annual Symposium on the Theoretical Aspects of Computer Science, Bordeaux, France, February 11-13, 1988*, vol. 294 of *Lecture Notes in Computer Science*, (pp. 61–72). Springer, Berlin.

UNGER, W. (1990). *Färbung von Kreissehnengraphen*. Ph.D. thesis, Universität Paderborn.

UNGER, W. (1992). The complexity of colouring circle graphs (Extended abstract). In *STACS '92: Proceedings of the 9th annual Symposium on Theoretical Aspects of Computer Science*, (pp. 389–400). Springer, London.

VAZIRANI, V. V. (2001). *Approximation Algorithms*. Springer, Berlin Heidelberg.

WAGNER, K. (1984). Monotonic coverings of finite sets. *Elektronische Informationsverarbeitung und Kybernetik*, *20*(12), 633–639.

WINTER, T. (2000). *Online and Real-Time Dispatching Problems*. Ph.D. thesis, Abteilung für Mathematische Optimierung, Technische Universität Carolo-Wilhelmina zu Braunschweig.

WINTER, T., & ZIMMERMANN, U. T. (2000). Real-time dispatch of trams in storage yards. *Annals of Operations Research*, *96*, 287–315.

WÖCKEL, F. (1949). Nahgüterzugbildung im Flachbahnhof durch Ablauf. *Eisenbahntechnik*, *1*, 5–12.

# Name Index

# Subject Index

# Nomenclature

$\oplus$ ..... concatenates two integer sequences
$\alpha$ ..... parameters for track topology
$\alpha(G)$ ..... independence number of graph $G$
$\alpha(i, j)$ ..... type of arc $(i, j)$
$\beta$ ..... parameters for sorting mode
$\gamma$ ..... parameters for structure of output sequence
$\delta$ ..... parameter for ITERATIVE ROUNDING
$\Delta$ ..... set of triangles
$\Delta_i$ ..... ($i$-th) triangle
$\lambda$ ..... vector of Lagrangian multipliers
$\pi_i$ ..... $i$-th element of permutation $\Pi$
$\Pi$ ..... permutation
$\chi(G)$ ..... chromatic number of graph $G$
$\omega(G)$ ..... clique number of graph $G$
$\mathcal{C}$ ..... set of chords
$\mathcal{C}_i$ ..... ($i$-th) chord
$\mathcal{E}$ ..... track execution order
$\mathcal{E}^c$ ..... cyclic track execution order
$\mathcal{F}$ ..... set of interval-filaments
$\mathcal{G}^S$ ..... set of integers contained in integer sequence $S$
$\mathcal{H}_i$ ..... $i$-th harmonic number
$\mathcal{I}$ ..... set of intervals on the real line
$\mathcal{I}_i$ ..... ($i$-th) interval
$\mathcal{I}^{\mathfrak{g}}$ ..... set of (time) intervals for all units of group $\mathfrak{g}$
$\mathcal{I}^{\mathcal{P}_i}$ ..... interval between $\mathfrak{r}_1^i$ and $\mathfrak{r}_{n_i^c}^i$
$\mathcal{NP}$ ..... class of non-deterministic polynomial solvable problems

$\mathcal{N}$ ..... network

$\mathcal{P}$ ..... set of polygons or class of polynomial solvable decision problems

$\mathcal{P}_i$ ..... ($i$-th) polygon

$\mathcal{T}$ ..... set of trapezoids

$\mathcal{V}$ ..... version of SRS

$\mathfrak{A}$ ..... algorithm

$\mathfrak{C}$ ..... set of cliques

$\mathfrak{I}$ ..... instance of a mathematical problem

$\mathfrak{p}(\mathfrak{r})$ ..... point on a circle that corresponds to real $\mathfrak{r}$

$\mathfrak{P}$ ..... mathematical problem

$\mathfrak{P}_\mathfrak{s}$ ..... subproblem of $\mathfrak{P}$

$\mathfrak{r}$ ..... real number

$\mathfrak{r}_j^i$ ..... $j$-th corner point of polygon $\mathcal{P}_i$

$\mathfrak{R}$ ..... set of reals

$\mathfrak{R}_i$ ..... set of reals that correspond to the corner points of polygon $\mathcal{P}_i$

$\mathfrak{X}$ ..... set of decision variables

$\mathscr{I}_0$ ..... information given in advance

$\mathscr{I}_p$ ..... information revealed by any input portion

$a_i$ ..... arrival time of unit $i$

$b$ ..... bound on track length (number of units)

$coG$ ..... complement graph of graph $G$

$C$ ..... clique

$C_{\max}$ ..... maximum clique

$C_{\mathrm{wmax}}$ ..... maximum weighted clique

$d_i$ ..... departure time of unit $i$

$E$ ..... set of edges

$f(h,t)$ ..... number of different realizable paths for $h$ humping steps and $t$ tracks

$g$ ..... number of integers (groups) in an integer (input) sequence

$g_{\hat{\mathfrak{o}}}$ ..... number of groups within outbound train $\hat{\mathfrak{o}}$

$G$ ..... graph

$G^{\mathcal{V}}$ ..... $\mathcal{V}$-graph
$G^{C}$ ..... circle graph
$G^{IF}$ ..... interval-filament graph
$G^{I}$ ..... interval graph
$G^{PC}$ ..... polygon-circle graph
$G^{PI}$ ..... point-interval graph
$G^{P}$ ..... permutation graph
$G^{T}$ ..... trapezoid graph
$G[V']$ ..... subgraph of $G$ induced by $V'$
$h$ ..... number of humping steps
$h^*$ ..... minimum number of required humping steps
$I$ ..... independent set
$I_{\max}$ ..... maximum independent set
$I_{\mathrm{wmax}}$ ..... maximum weighted independent set
$m$ ..... number of edges
$M$ ..... set or large constant
$n$ ..... number of elements (either in a sequence or in a set)
$n^c$ ..... total number of corner points of polygons in $\mathcal{P}$
$n_i^c$ ..... number of corner points of polygon $\mathcal{P}_i$
$n^{\mathcal{P}}$ ..... number of polygons in $\mathcal{P}$
$n_{\hat{\mathfrak{o}}}$ ..... number of units in outbound train $\hat{\mathfrak{o}}$
$\bar{n}_{\mathfrak{g}}$ ..... absolute frequency of integer (group) $\mathfrak{g}$ in an integer (input) sequence
$N_G(v)$ ..... neighborhood of vertex $v$ in graph $G$
$o$ ..... number of outbound trains
$p_{\mathfrak{g}}^f$ ..... first position of integer (group) $\mathfrak{g}$ in an integer (input) sequence
$p_{\mathfrak{g}}^l$ ..... last position of integer (group) $\mathfrak{g}$ in an integer (input) sequence
$P_{\mathfrak{g}}$ ..... period of integer (group) $\mathfrak{g}$
$P_S$ ..... partition of integer sequence $S$
$P(S)$ ..... set of periods for all integers (groups) contained in $S$
$r$ ..... number of railcar moves

$r^*$ ..... minimum number of required railcar moves
$s_i$ ..... $i$-th element (unit) of integer (input) sequence $S$
$S$ ..... integer sequence
$S^{\leftrightarrow}$ ..... element reversing sequence of integer sequence $S$
$S^{\updownarrow}$ ..... integer reversing sequence of integer sequence $S$
$S^{\mathcal{I}}$ ..... integer sequence according to the set of time intervals $\mathcal{I}$
$S^{\hat{\mathfrak{o}}}$ ..... subsequence of input sequence containing all units of outbound train $\hat{\mathfrak{o}}$
$S_{n,g}$ ..... integer sequence with $n$ elements that contains $g$ different integers
$S^{c}_{n,g}$ ..... cyclic integer sequence
$S^{\text{out}}$ ..... output sequence
$S^{\text{out}}_{\hat{\mathfrak{o}}}$ ..... output sequence of outbound train $\hat{\mathfrak{o}}$
$t$ ..... number of tracks
$t^*$ ..... minimum number of required tracks
$V$ ..... set of vertices
$x$ ..... variable in mathematical program
$X$ ..... feasible region (set of solutions)
$X_{\mathfrak{s}}$ ..... feasible region of subproblem $\mathfrak{P}_{\mathfrak{s}}$
$y$ ..... variable in mathematical program
$z$ ..... objective value
$z^*$ ..... optimal objective value
$z^{\text{LD}}$ ..... objective value of a Lagrangian dual
$z^{\text{LP}}$ ..... objective value of an LP relaxation
$z^{\text{LR}}$ ..... objective value of a Lagrangian relaxation

# List of Figures

# List of Algorithms

# List of Tables

# Lebenslauf

## Persönliche Daten

Ronny Stefan Hansmann
geboren am 26. April 1978 in Rabenstein
ledig

## Bildungsgang

| | |
|---|---|
| 1984 – 1992 | Grundschule in Limbach-Oberfrohna |
| 1992 – 1996 | Gymnasium *Albert-Schweitzer* in Limbach-Oberfrohna,<br>Abitur mit der Note 1,8 |
| 1997 – 2003 | Studium der Angewandten Mathematik<br>mit Nebenfächern Informatik und Betriebswirtschaftslehre<br>an der TU *Bergakademie* zu Freiberg,<br>Diplom mit der Note "*Sehr Gut*" |

## Beschäftigungszeiten

| | |
|---|---|
| 1996 – 1997 | Wehrdienst |
| 2002 – 2003 | Studentische Hilfskraft |
| 2004 – dato | Wissenschaftlicher Mitarbeiter<br>an der TU *Carolo-Wilhelmina* zu Braunschweig<br>bei Prof. Dr. Uwe T. Zimmermann |